Search-Based Genetic Optimization for Deployment and Reconfiguration of Software in the Cloud

Sören Frey, Florian Fittkau, and Wilhelm Hasselbring
Software Engineering Group
Kiel University
24118 Kiel, Germany
{sfr, ffi, wha}@informatik.uni-kiel.de

Abstract—Migrating existing enterprise software to cloud platforms involves the comparison of competing cloud deployment options (CDOs). A CDO comprises a combination of 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 extremely huge. 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 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 systems [2]. However, 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. Then, 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.

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. All of those design decisions form a multitude of cloud deployment options (CDOs) that 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 [3]. Furthermore, as we also intend to support distributed systems, we consider the QoS-aware composition of software components that run on one node as a single service that is provided to an arbitrary number of components on other nodes. Such deployment optimization problems are intractable as they are known to be NP-hard [4]. In our previous work, we introduced the simulation tool CDOSim [5] that implements a phase of our cloud migration approach CloudMIG [6, 7]. 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 manually configured and simulated on the basis of a reverse-engineered architectural system model with monitored or synthetic workload. However, the design space that spans for all possible CDOs is huge, the elements of a single CDO exhibit complex non-linear interdependencies, and CDO simulation runs are very 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 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.

In this paper, 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 [8]. CDOXplorer is implemented in our tool CloudMIG Xpress and supports IaaS-based cloud environments [9], 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 purposes as well. By incorporating CDOSim, CDOXplorer is a member of the simulation-based optimization class [10].

Here, the evaluation of the used fitness function is, in contrast to most 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. 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 almost all 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%.

CloudMIG Xpress and our experiment code and data are available online as open source software such that interested researchers may repeat or extend our experiments.1

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

II. MOTIVATING EXAMPLE

We consider the example deployment of a software system that is shown on the left side of Figure 1. This system should be moved 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

1http://www.cloudmig.org

status-quo assignment of components to physical machines.

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

As mentioned earlier, 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.

Thus, the three status-quo nodes and the deployed components shown in the left part of Figure 1 result in the three services Service 0 to Service 2 in the right part of Figure 1 that exhibit a similar assignment of components. As can be seen in the lower right part of Figure 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. For example, at the time of writing, Amazon EC2’s m1.large VM instance type provides 7.5 GB memory and two virtual cores that together provide approximately the CPU capacity of four 1.7 GHz Xeon processors from 2006. The remaining Service 0 in Figure 1 is deployed to an own VM that builds upon the m1.medium VM instance type.

The basic CDO of Figure 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. Furthermore, up to this point, 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, 10 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 2,744,000 CDO candidates. Without using potent heuristics, all CDOs would have to be simulated for reliably finding competitive solutions.
III. INPUT MODELS

In this section, we briefly describe the four input models that have to be provided to CDOXplorer so it can find well-suited cloud deployment models and reconfiguration rules. We provide tool support for creating these models. The architectural model, status-quo deployment model, workload profile, and cloud profile are described in the following.

An architectural 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 [11]. CloudMIG Xpress generates architectural models that conform to the ISO/IEC 19506 standard Knowledge Discovery Meta-Model [12] (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 mega integer plus instructions per second (MIPIPS) [5]. 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. Currently, we support Kieker [13] 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’ performance capabilities. MIPIPS can also be measured by executing the mentioned benchmark for each VM instance type of a cloud environment. We plan to develop and integrate a public repository such that cloud profiles can be exchanged.

IV. OUTPUT MODELS

This section explains the result models that are included in a CDO as delivered by CDOXplorer. The CDOs specify a cloud deployment model and a set of reconfiguration rule models. In IaaS-based cloud environments, resources can be dynamically acquired and released by executing reconfiguration actions to counteract under- and over-provisioning and to ensure the compliance with specified SLAs. Two reconfiguration actions together define one of the scaling types vertical scaling or horizontal scaling.

Considering vertical scaling, the reconfiguration actions scale-up and scale-down are available. Scaling up adds more resources to a VM, e.g., a further CPU or more memory, whereas scaling down removes resources. Horizontal scaling employs VMs that use the same VM instance type. VM instances are added (scale-out) or shut down (scale-in).

We now describe the basic structure of CDOs that is shown in Figure 2. A Cloud Deployment Option refers to a single Cloud Environment, that is derived from a cloud profile, and contains so-called Node Configurations. A Node Configuration describes specifics of a VM instance. For example, an included Service Composition container refers to the Services that are deployed on this VM. To link Services with the architectural input model, they reference parts of the previously 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 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 VMs that have to be present and refer to a Grow Action or Shrink Action that define the reconfiguration actions that are used for scaling. A combination of two actions has to comply with the previously explained scaling types.

When applying vertical scaling, the mipipsMultiple attribute of Grow Rules and Shrink Rules becomes relevant. It implicitly specifies the VM instance type of the newly started VM, which is given by multiplying the MIPIPS value of the current VM with mipipsMultiple and rounding to the nearest MIPIPS value of any (the intended) VM instance type. A Condition constitutes a trigger for executing the reconfiguration action with the help of a CPU utilization threshold and a time period. For example, concerning a scale-out action, an additional VM could be started when the CPU utilization is above 80% for at least 20 minutes. The Scope element would then define if the 80% refers to the specific VM or to the average of all VMs that were started from the corresponding Node Configuration.

V. OUR SIMULATION-BASED GENETIC ALGORITHM

This section details our simulation-based genetic algorithm CDOXplorer. First, Section V-A describes its basic design. Then, Section V-B formalizes the optimization problem that is tackled by CDOXplorer. Sections V-C and V-D detail the crossover and mutation operator of CDOXplorer, respectively.

A. Basic 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 a CDO.

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 generation includes the following four basic steps S1-S4 [8]:

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 [14] 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 V-C and V-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, we had to limit the population size and configured the basic parameters of CDOXplorer as follows. Our populations contain 50 individuals (population size $\alpha$), 25 individuals are selected from each generation for reproduction (number of parents $\mu$) and each generation spawns 50 children (number of children $\lambda$). 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 Figure 2, 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. Figure 3 illustrates the basic chromosomes and genes that are processed by CDOXplorer.

The chromosomes correspond to the class structure of Figure 2 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 Figure 2. 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 points in Figure 3 are detailed in Section V-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 Figure 2. We limited their values to a narrow range and discrete spaces for avoiding a further growth of the search-space.

Figure 4 presents three examples of CDOs that are encoded as genotypes. CDO1 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 Figure 3. 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. The second node configuration that includes a reconfiguration rule is represented by the genes 5 (SE) - 20 (S5). The further example CDO2 also includes two node configurations and uses Amazon EC2, whereas CDO3 shows a genotype using the cloud environment Microsoft Windows Azure and only one node configuration. We will refer to these genotypes later.

B. Problem Statement

After introducing the basic structure of our genetic algorithm, we can describe the multi-objective optimization problem that is tackled by CDOXplorer as follows. Let $\Phi$ 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 $\text{costs}(x)$, $\text{rt}(x)$, and $\text{sla}(x)$ that determine the costs, average response
### TABLE I: Design of the used genes.

<table>
<thead>
<tr>
<th>Gene Range</th>
<th>Description</th>
<th>Chromosome</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE N</td>
<td>Cloud environment id</td>
<td>Cloud Env.</td>
</tr>
<tr>
<td>SE N</td>
<td>Service id</td>
<td>Service Comp.</td>
</tr>
<tr>
<td>IT N</td>
<td>VM Instance type id</td>
<td>Initial Start C.</td>
</tr>
<tr>
<td>N1 N</td>
<td>Nr. of VM instances to start initially</td>
<td>Initial Start C.</td>
</tr>
<tr>
<td>GA 0.1</td>
<td>Grow action: 0; scale-up, 1; scale-out</td>
<td>Grow Rule</td>
</tr>
<tr>
<td>G1 N</td>
<td>Minimum nr. of VM instances</td>
<td>Grow Rule</td>
</tr>
<tr>
<td>G2 1.1-3.0</td>
<td>MIPPS multiple in steps of 0.1</td>
<td>Grow Rule</td>
</tr>
<tr>
<td>G3 0.1</td>
<td>Condition scope: 0; single VM, 1: all VMs</td>
<td>Grow Rule</td>
</tr>
<tr>
<td>G4 0.05-1.0</td>
<td>Condition median utilization in steps of 0.05</td>
<td>Grow Rule</td>
</tr>
<tr>
<td>S1 N</td>
<td>Minimum nr. of VM instances</td>
<td>Shrink Rule</td>
</tr>
<tr>
<td>S2 0.1-0.9</td>
<td>MIPPS multiple in steps of 0.1</td>
<td>Shrink Rule</td>
</tr>
<tr>
<td>S3 0.1</td>
<td>Condition scope: 0; single VM, 1: all VMs</td>
<td>Shrink Rule</td>
</tr>
<tr>
<td>S4 0.0-0.95</td>
<td>Condition median utilization in steps of 0.05</td>
<td>Shrink Rule</td>
</tr>
<tr>
<td>S5 5-60</td>
<td>Condition time period in steps of 5 minutes</td>
<td>Shrink Rule</td>
</tr>
</tbody>
</table>

---

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 respect to the constraints that are described 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.

$$y \prec z : y \text{ is contained in } z$$

$$\Delta(y, z) : \text{Number of } y \text{ in } z$$

$(T)$: Set of VM instance types of cloud environment $c$

$x$ has to comply with the following constraints $(1) - (7)$.

1. $\forall s \in S, x \in \Phi : \Delta(s, x) \geq 1$
2. $\forall s \in S, n \in N : \Delta(s, n) \leq 1$
3. $\forall n \in N : \exists s \in S, s < n$
4. $\forall IT < x, CE < x, x \in \Phi : IT \in T_{CE}$
5. $\forall n \in N : \Delta(gr, n) = \Delta(sr, n) \leq 1$
6. $\forall gr < n, sr < n, n \in N : gr > GA = SA < sr$
7. $\forall gr < n, sr < n, n \in N : gr > G4 > SA < sr$

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)), 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)).

### C. Crossover Operator

The reproduction procedure involves 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 of the parents to their offspring. As the simulation of CDOs is very expensive, we designed the crossover operator to produce only feasible individuals (see Section V-B). Hence, we restricted the mixing to dedicated positions in the genotype called crossover points. We defined the four crossover points CP1–CP4 that are shown in Figure 3. They are selected by chance, are aligned to the chromosomes’ boundaries, and specify corresponding gene sequences that can be swapped.

Figure 5 shows an example for applying the crossover operator with the help of the previously introduced CDOs of Figure 4. The example considers CDO1 and CDO2. Both CDOs include two node configurations. As CP2 was selected, both initial start configurations of CDO1 and CDO2 are swapped.

### 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 can be seen in Figure 3. As a mutation also has to maintain the inner structure of a chromosome, we divided the mutation operator in five sub operators that are described in the following.

- **M-CE** Mutates the 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 VMs that are initially started for
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 and vice versa.

VI. EXPERIMENTAL EVALUATION

To evaluate our simulation-based genetic algorithm CDO-
Xplorer, we implemented it as a component of our open source
tool CloudMIG Xpress. We let CDOXplorer create CDO can-
didates 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?

RQ3: Scalability—Is CDOXplorer applicable for both sin-
gle and multi cloud environment scenarios?

CDOXplorer utilizes our simulator CDOSim. Please note that
the following evaluation covers only CDOXplorer. For an
evaluation of CDOSim that demonstrates its applicability and
precision, we refer to our previous work [5, 15]. We begin with
explaining the methodology that we used to tackle the research
questions in Section VI-A. The experimental setting is then
described in Section VI-B. The research questions RQ1-RQ3 are
detailed in Sections VI-C to VI-E, respectively, before the
threats to validity are discussed in Section VI-F.

A. Methodology

We used the Opt4J framework for meta-heuristic optimiza-
tion [16] as a basis for our implementation of CDOXplorer
and also applied the NSGA-II algorithm for the selection of
parent individuals (see Section V-A). As NSGA-II utilizes
so-called fronts of non-dominated individuals [14], a pareto
optimum is also called pareto optimal front in this context.
There exist several standard performance metrics that basic-
ally 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 [17], we decided against using a single
metric. Instead, we applied the two popular metrics inverted
generational distance (IGD) [18] and hypervolume indicator
(HV) [19] that are illustrated in Figure 6. Usually, the metrics
specify the approximation towards the true pareto optimal
front (PFtrue) if this is known. In our case, PFtrue 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
OPFBest. Another pareto optimal front depicted in Figure 6
is PFKnown. It is the result of each execution of CDOXplorer,
whereas each execution includes 3,000 CDOSim simulations.
The cube’s axes in Figure 6 indicate the three objectives that
are optimized by our algorithm: costs, response times, and
SLA violations. Starting from the individuals in OPFBest,
the nearest individuals in PFKnown, in terms of the Euclidean
distance, are used for calculating the IGD metric. This metric
directly measures the distance from OPFBest to PFKnown.
Hence, smaller values indicate a better approximation.

In contrast, the values produced by the HV metric be-
come bigger as the approximations become better. HV mea-
ures the volume in hyper-dimensional space that is covered
by PFKnown regarding a reference point r ∈ Rd in d-
dimensional space. As we consider three objectives and con-
figure HV accordingly, the mentioned volume corresponds to
the volume in three-dimensional space that is colored blue in
the illustration of Figure 6. HV therefore implicitly rates the
distance from OPFBest to PFKnown, as is also done by IGD.
Additionally, both metrics assess the spread of PFKnown,
that constitutes another important quality characteristic. When
covering larger parts of the search space, there exist less un-
explored 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 opti-
mization 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 [20].
Therefore, we use the simple yet effective stochastic algo-
rithms simple random sampling (SI-RS) and systematic random
sampling (SY-RS) [21] 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 can iterate over
cd for the 0, floor(3,000) interval at the beginning of

Fig. 6: The metrics inverted generational distance and hypervolume
indicator (colored blue) are used for evaluation.
CDOXplorer uses 60 generations with populations of 50 individuals. Hence, CDOXplorer applies 3,000 simulations in a single run. The runs for CDOXplorer, SI-RA, SY-RA, 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 the single and multi cloud scenario that were used to approximate two pareto optimal fronts that were set to \( OPF_{Best} \) for evaluating the respective scenarios. The first of these two pareto optimal fronts is shown in Figure 8.

B. Experimental Setting

For incorporating public clouds, we measured the MIPIPS values (see Section III) 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.

- \( SC_S \): Amazon EC2
- \( SC_M \): Amazon EC2, Microsoft Windows Azure, Eucalyptus

We extracted a KDM model from the open source ERP system Apache OBiz 10.04.\(^2\) Then, we deployed this system 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 of 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 \( 2\% \). For Eucalyptus, we defined a synthetic cost model where the prices for VMs follow the capabilities of our VM instance types.

\(^2\)http://ofbiz.apache.org/

D. RQ2: Competitiveness

This research question addresses the competitiveness with other state-of-the-art search approaches by comparing CDOXplorer with SI-RA, SY-RA, and SI-AN. Table II lists the results for the metrics IGD and HV for the single cloud scenario \( SC_S \). Table III shows these results for \( SC_M \). The tables list the
TABLE II: Search Over A Single Cloud Profile ($SC_S$).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Search Method</th>
<th>CDOXplorer</th>
<th>SI-RS</th>
<th>SY-RS</th>
<th>SI-AN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>2.70E-02</td>
<td>3.67E-02</td>
<td>4.11E-02</td>
<td>3.28E-02</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>2.10E-03</td>
<td>2.13E-03</td>
<td>3.61E-03</td>
<td>2.85E-03</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>2.72E-02</td>
<td>3.65E-02</td>
<td>4.21E-02</td>
<td>3.20E-02</td>
<td></td>
</tr>
<tr>
<td>Min (best)</td>
<td>2.16E-02</td>
<td>3.34E-02</td>
<td>3.40E-02</td>
<td>2.76E-02</td>
<td></td>
</tr>
<tr>
<td>Max (worst)</td>
<td>3.03E-02</td>
<td>4.07E-02</td>
<td>4.83E-02</td>
<td>3.95E-02</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Search Method</th>
<th>CDOXplorer</th>
<th>SI-RS</th>
<th>SY-RS</th>
<th>SI-AN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>4.48E-01</td>
<td>4.14E-01</td>
<td>4.41E-01</td>
<td>4.44E-01</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>2.08E-03</td>
<td>1.96E-03</td>
<td>2.89E-03</td>
<td>2.09E-03</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>4.48E-01</td>
<td>4.40E-01</td>
<td>4.41E-01</td>
<td>4.44E-01</td>
<td></td>
</tr>
<tr>
<td>Min (worst)</td>
<td>4.44E-01</td>
<td>4.36E-01</td>
<td>4.35E-01</td>
<td>4.40E-01</td>
<td></td>
</tr>
<tr>
<td>Max (best)</td>
<td>4.54E-01</td>
<td>4.46E-01</td>
<td>4.46E-01</td>
<td>4.48E-01</td>
<td></td>
</tr>
</tbody>
</table>

TABLE III: Search Over Three Cloud Profiles ($SC_M$).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Search Method</th>
<th>CDOXplorer</th>
<th>SI-RS</th>
<th>SY-RS</th>
<th>SI-AN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>3.08E-02</td>
<td>7.18E-02</td>
<td>8.03E-02</td>
<td>3.37E-02</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>4.88E-03</td>
<td>2.78E-03</td>
<td>4.41E-03</td>
<td>4.50E-03</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>3.12E-02</td>
<td>7.17E-02</td>
<td>7.90E-02</td>
<td>3.38E-02</td>
<td></td>
</tr>
<tr>
<td>Min (best)</td>
<td>2.13E-02</td>
<td>6.50E-02</td>
<td>7.23E-02</td>
<td>2.52E-02</td>
<td></td>
</tr>
<tr>
<td>Max (worst)</td>
<td>4.16E-02</td>
<td>7.76E-02</td>
<td>8.88E-02</td>
<td>4.67E-02</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Search Method</th>
<th>CDOXplorer</th>
<th>SI-RS</th>
<th>SY-RS</th>
<th>SI-AN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5.65E-01</td>
<td>5.20E-01</td>
<td>5.18E-01</td>
<td>5.63E-01</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>1.82E-03</td>
<td>1.95E-03</td>
<td>2.43E-03</td>
<td>1.68E-03</td>
<td></td>
</tr>
<tr>
<td>Median</td>
<td>5.65E-01</td>
<td>5.20E-01</td>
<td>5.17E-01</td>
<td>5.63E-01</td>
<td></td>
</tr>
<tr>
<td>Min (worst)</td>
<td>5.61E-01</td>
<td>5.16E-01</td>
<td>5.13E-01</td>
<td>5.61E-01</td>
<td></td>
</tr>
<tr>
<td>Max (best)</td>
<td>5.70E-01</td>
<td>5.25E-01</td>
<td>5.25E-01</td>
<td>5.68E-01</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9: CDOXplorer advantage relative to other approaches.

is useful to compare these observations with those from the other search methods. CDOXplorer provides in all cases the best results. Considering SI-RS and SY-RS, their values for the IGD metric grow even more and nearly doubled when using $SC_M$. SI-AN also suffers from a deterioration when transitioning from the $SC_S$ to the $SC_M$ scenario. Though, the IGD values increase less. The values for HV show a similar development. CDOXplorer is better than all other search methods and the scalability is better compared with SI-RS and SY-RS, but SI-AN exhibits a slightly higher improvement.

In summary, we conclude that (1) CDOXplorer scales better than SI-RS and SY-RS, but (2) slightly worse than SI-AN. Although CDOXplorer performs better in all cases in terms of the absolute median values, we will conduct additional experiments in our future work to further investigate scalability.

E. Threats to Validity

There are several issues that form 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, as indicated by the results of Section VI-E, scalability is 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 strived after 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. However, we plan to address the stated limitations in our future work.

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 in the light that 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 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 $SC_M$.

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 ($\alpha = 0.016$) for multiple comparisons, $H_0$ is rejected with significance level ($\alpha$) 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. Figure 9 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.

E. RQ3: Scalability

This research question analyzes CDOXplorer’s scalability. That means, if CDOXplorer can retain its performance when more cloud profiles are considered for providing CDO candidates or if a potential performance degradation is still acceptable. This is of particular interest as the number of generations and the population size remain stable but the search-space size grows linearly with each new cloud profile.

As described in Section VI-C, the value for the coefficient of variation (with IGD) and the value for the IGD metric itself grows when CDOXplorer processes $SC_M$. However, the HV metric value and the coefficient of variation even become better, when more cloud profiles are used. Furthermore, it
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.

VII. 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.

**Deployment optimization in non-cloud scenarios.** For improving the deployment architecture of distributed systems regarding arbitrary QoS properties, Malek et al. [23] 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 a genetic algorithm. The utility functions are used as fitness functions, whereas CDOXplorer uses simulation runs to obtain fitness values. Furthermore, Malek et al. [23] do not consider the integration of reverse-engineered code models and monitored workload, as is supported by our approach.

Martens et al. [24] focus on component-based systems and at finding optimized software and deployment 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 PerOpterxiv provides tool support and explores four degrees of freedom, e.g., processor speeds. No dynamic resource scaling is 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. [25]. 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. 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.

Zhang et al. [26] 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.

**Non-evolutionary cloud deployment optimization.** As with our approach CloudMIG, Wu et al. [27] consider resource usage optimization for SaaS providers that build upon leased VMs. Wu et al. 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. do not consider distributed applications, runtime reconfiguration, and arbitrary workload.

Trummer et al. [28] 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. serve the same purpose as our status-quo deployment models.

San Aniceto et al. [29] 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. These are frequently offered 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. [30]. 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.

**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., [31, 32]). In contrast, Wada et al. [33] follow, in common with CloudMIG, a cloud user perspective and contribute the genetic algorithm E³-R that explores deployment configurations for optimizing services’ QoS attributes. E³-R can also reduce redundant QoS objectives and estimate the performance of configurations using queueing theory and historic mean arrival rates. However, E³-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. [34]. 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.
We presented our simulation-based genetic algorithm CDOXplorer that optimizes cloud deployment options (CDOs) for supporting the migration of software to the cloud. A CDO comprises a deployment architecture and runtime reconfiguration rules that can start further VM instances when the overall utilization exceeds a given threshold, for instance. 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 provider 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.

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. Our evaluation showed that CDOXplorer 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.

ACKNOWLEDGMENTS

The authors would like to thank Guido Scherp for his support in setting up the experiments.

REFERENCES