Kalpa Publications in Computing

Volume 9, 2018, Pages 1–14

LPAR-22 Workshop and Short Paper Proceedings

Influence of Variables Encoding and Symmetry Breaking on the Performance of Optimization Modulo Theories Tools Applied to Cloud Resource Selection*

Mădălina Eraşcu[1,2], Flavia Micota[1,2], and Daniela Zaharie[1,2]

[1] Department of Computer Science, West University of Timişoara, Romania
[2] Institute e-Austria Timişoara, Romania
{madalina.erascu,flavia.micota,daniela.zaharie}@e-uvt.ro

Abstract

The problem of Cloud resource provisioning for component-based applications is very important. It consists in the allocation of virtual machines (VMs) from various Cloud Providers (CPs), to a set of applications such that the constraints induced by the interactions between components and by the components hardware/software requirements are satisfied and the performance objectives are optimized (e.g. costs are minimized). It can be formulated as a constrained optimization problem and tackled by state-of-the-art optimization modulo theories (OMT) tools. The performance of the OMT tools is highly dependent on the way the problem is formalized as this determines the size of the search space. In the case when the number of VMs offers is large, a naive encoding which does not exploit the symmetries of the underlying problem leads to a huge search space making the optimization problem intractable. We overcame this issue by reducing the search space by using: *(1)* a *heuristic* which exploits the particularities of the application by detecting cliques in the conflict graph of the application components fixing all components of the clique with the largest number of component instances, and *(2)* a *lex-leader method* for breaking variable symmetry where the canonical solution fulfills an order based on either the number of components deployed on VMs, or on the VMs price. As the result, the running time of the optimization problem improves considerably and the optimization problem scales up to hundreds of VM offers. We also observed that by combining the heuristic with the lex-leader method we obtained better computational results than by using them separately, suggesting the fact that symmetry breaking constraints have the advantage of interacting well with the search heuristic being used.

1 Introduction

In an article published in February 2017, Gartner[1] stated that, due to digital market demand, the successful application service providers are "investing in platforms that allow *component-based development* based on converged infrastructure as a service (IaaS) and platform as a

*This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS/CCCDI - UEFISCDI, project number PN-III-P2-2.1-PED-2016-0550, within PNCDI III.

[1] https://www.gartner.com/doc/3624017/market-trends-application-service-providers

G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe and M. Veanes (eds.), LPAR-IWIL 2018 (Kalpa Publications in Computing, vol. 9), pp. 1–14

service (PaaS) (Amazon Web Services, Azure, Force.com) [...]" thus proving the wide adoption of component-based software development. Other advantages of this paradigm include code reusability and short application creation and delivery time.

With the wide adoption of Cloud Computing, application components are deployed on VMs which are offered by various CPs: Amazon Web Services (https://aws.amazon.com/), Google Cloud (https://cloud.google.com/), Azure (https://azure.microsoft.com/en-us/), etc. One of the natural questions arising is: *Which CPs offer the best infrastructure at a fair budget such that my application performance requirements are fulfilled?*

To answer the question one must solve a *resource management problem*, that is: *(1) mapping* the components to VMs such that the computing, storage, networking requirements of the application are fulfilled, and *(2) selection* of VMs offers which minimize the cost.

In the framework of the MANeUveR project[2], we are approaching the problem by constructing a recommendation engine which translates the problem into a constrained optimization one aiming to minimize the leasing cost of all VMs needed for deployment such that the hardware and interaction constraints of the application components are fulfilled.

The importance of such a recommendation engine is motivated by the fact that there are various industrial initiatives, e.g. Cloud Foundry (http://www.cloudfoundry.com/), Jujucharm (https://jujucharms.com/) assisting the user in the application Cloud deployment process but which require the knowledge of a preliminary mapping of the services and packages to locations (VMs) such that all the requirements are satisfied. This is typically solved by custom scripts and manual techniques requiring human intervention, hence the process is susceptible to errors.

Constrained optimization problems have been previously solved using: *(1) exact*: e.g. *constraint programming* [7] and *SMT solving* [1]; and *(2) approximate* [10] approaches.

In a recent work [12], we investigated how the three approaches enumerated above scale in practice. More precisely:

- we introduced the formalization of the problem, briefly presented in Section 3;
- we applied the constraint programming solver OR-tools (https://developers.google.com/optimization/) and the OMT solver νZ for solving the optimization problem for the three case studies Oryx2, Secure Web Container and Wordpress (no heuristics used);
- since they did not scale for more than a few dozens of VMs offers, which is not a realistic number in applications, we designed a population-based metaheuristic algorithm that uses a problem tailored mutation and a simple one-to-one selection.

The results showed that the metaheuristic provides acceptable solutions, however not necessarily optimal, for hundreds of offers. The SMT approach gave better timings than the metaheuristic for problems involving low number of components and VM offers.

In this paper, we address the computational issues behind the adoption of state-of-the-art OMT solvers for solving constrained optimization problems arising in the Cloud resource management domain.

The main contributions of the paper are:

- SMT encoding of the optimization problem, in particular the link between application components hardware requirements and CPs offers (Section 3.2);
- investigation of the scalability of the state-of-the-art OMT tools, namely νZ [4] and OptiMathSAT [13] (Section 4.1);
- proposal of: *(1)* a *heuristic* tailored for component-based applications which exploits the possible cliques in the conflict graph of the components, and *(2)* a *lex-leader method* for

[2] https://merascu.github.io/links/MANeUveR.html

breaking variable symmetries where the canonical solution satisfies an order based on, either the number of components deployed on VMs, or on the VMs price (Section 4.2).
- extension of test problems benchmarks with examples coming from the Cloud deployment scenarios of complex component-based applications.

The paper is structured as follows. We continue this section with related work. In Section 3 we present the formalization of the problem as a linear programming problem. Section 4 contains comparative results of the OMT solvers and proposes two strategies for improving their scalability. Section 5 concludes the paper and presents future research directions.

Related Work. Optimization problems arising in Cloud Computing are typically solved using approximate approaches (evolutionary algorithms) because of their complexity and the requirement of low computational time. However, these methods are suboptimal and there are no theoretical results which allow to estimate how far from the real optimum the solution is. The benefit of exact techniques is that they guarantee the optimal solution, with the disadvantage of higher computational time.

To the best of our knowledge there is only the Zephyrus2 tool [1] using an SMT solver which addresses a problem similar to ours. The problem solved is that, given the application description (interaction constraints and hardware requirements) and VMs specifications, the aim is to minimize the cost and then the number of VMs leased for hosting the application. For solving it, constraint programming solvers Chuffed (`https://github.com/chuffed/chuffed`), Gecode (`https://github.com/Gecode/gecode`), OR-Tools and the SMT solver Z3 (`https://github.com/Z3Prover/z3`) were used. The paper presents various comparisons on how the tools perform on different types of constraints (linear/nonlinear), the conclusion being that the SMT solver performs better for nonlinear constraints while is outperformed by constraint programming solvers in the linear case.

Regarding the similarities/dissimilarities to our approach, we mention the following.
- In Zephyrus2, the list of offers is limited to four types of VMs; in our approach we considered up to 500 since this is a number more realistically expressing the number of current CPs offers. Moreover, in Zephyrus, the list of VM offers has to be specified by the user at the moment of application description, while in our case this list is obtained dynamically (via crawling).
- In Zephyrus2, according to the provided examples, only memory and price requirements are specified in the problem description but their approach can be easily extended to other types of constraints. The extension would increase the computational time.
- In Zephyrus2, the interaction between components is specified through logical expressions which would require the user of the application specialized knowledge. We avoid this by having JSON names for every constraint type we consider.
- In Zephyrus2, as symmetry breaking constraints, two strategies were considered: *(a)* an order among VMs (components are deployed first on the cheapest VMs) and, *(b)* components (lexicographic order based on the number of components on a VM) was considered. We included in our tests from Table 2 only the second one (encoded as VML) as the first one gave worse timings.
- Trimming heuristics for VMs were not considered in our approach, but they appear in Zephyrus2 with the outcome that they are outperformed by symmetry breaking constraints.
- A feature of Zephyrus2, which we do not consider, is the binding between collocated components. This would avoid configurations in which a component C_i uses the functionalities of another component C_j deployed on another VM while it could have used the

functionalities of the same component C_j deployed on the same VM as C_i.

SMT solvers have been also applied to other important problems in Cloud Computing. For example, in [6], SMT and constraint programming approaches have been applied to the problem of *cloud service selection*. More precisely, their framework automatically detects, in a first step, conflicts in enterprise policies and inconsistencies in user-defined requirements, and, if it is the case, generates explanations identifying problematic user requirements. Next, cloud services satisfying all enterprise policies and user requirements are selected. The framework has been applied to applications managing heterogeneous cloud infrastructure services in large enterprises. Paper [3] presents a different type of allocation than ours, namely *virtual data center*, that is finding an allocation of VMs to servers and links in the virtual network to links in the physical network. The allocation is valid if it fulfills the compute, memory, and network bandwidth requirements of each VM across the entire data center infrastructure, including servers, top-of-rack switches, and aggregation switches. The problem can be formulated as a SAT modulo monotonic predicates and solved using MonoSAT SMT solver [2].

2 Motivating Case Studies

In order to illustrate the applicability of our approach we consider three case studies[3].

Secure Web Container is an important web security application providing:

- *resilience* to attacks and failures, by introducing redundancy and diversity techniques, and
- protection from unauthorized and potentially dangerous accesses, by integrating proper *intrusion detection* tools.

We extended its formulation from [5] by including hardware requirements for its components and by formulating the mapping task (assignment of components to VMs) as a constrained optimization problem aiming to minimize the overall cost.

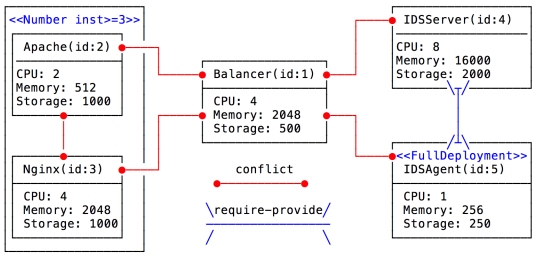

Figure 1: Secure Web Container Application

Wordpress open-source application is frequently used in creating websites, blogs and applications. For large businesses, which need multiple Wordpress websites, it can become difficult to manage all of them at the same instance, hence more instances are needed to be deployed. We chose this case study in order to compare our approach to Zephyrus and Zephyrus2 deployment tools [7, 1].

Oryx2 application is a realization of the lambda architecture, hence it is used in data analysis, and deploys the latest technologies such as Apache Spark (https://spark.apache.org/)

[3] Complete description of the case studies is at https://github.com/Maneuver-PED/ICCP2018.

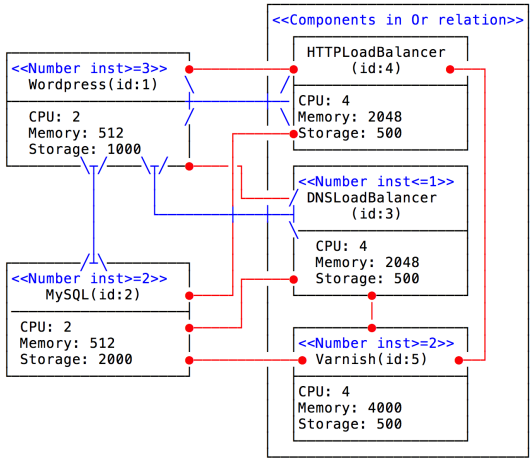

Figure 2: Wordpress Application

and Apache Kafka (https://kafka.apache.org/). It has a significant number of components interacting with each other and is highly used in practical applications.

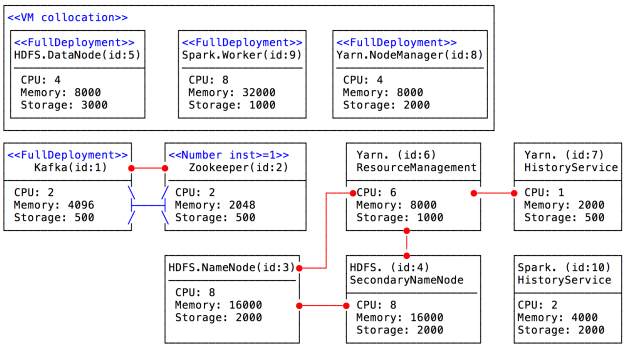

Figure 3: Oryx2 Application

These use cases have been selected such that they incorporate a fairly large set of constraints types. Other types of constraints might be easily included as long as they can be formulated using linear (in)equalities.

3 Problem Description

The problem to be solved is to find a mapping (of components to VMs) which: *(1)* satisfies the constraints induced by the interactions between components; *(2)* satisfies the hardware requirements of all components, and *(3)* minimizes the purchasing price. The formalization of the interaction constraints between components was introduced in [12] and briefly discussed here, while the link between the components hardware constraints and CPs offers was more challenging and it will be presented in detail.

In order to describe the problem in a more formal way, let us consider a set of N interacting component types, $\{C_1, C_2, \ldots, C_N\}$, to be assigned to a set of maximum M virtual machines, $\{V_1, V_2, \ldots, V_M\}$. Each component C_i is characterized by a set of L requirements concerning the hardware resources HR_t^i with $t = \overline{1, L}$. Each V_k is characterized by a leasing price $VP_k \in \mathbb{R}_+$. The problem can be formulated as a mathematical optimization problem, aiming:

1. to find a mapping between components and VMs such that all constraints are satisfied; the mapping is encoded like a set of binary variables, $a_{ik} \in \{0, 1\}$ for $i = \overline{1, N}$, $k = \overline{1, M}$, interpreted as follows: a_{ik} is 1 if C_i is assigned to V_k, and it is 0 otherwise.
2. to minimize the leasing price $\sum_{k=1}^{M} VP_k$. Note that some VP_k could be 0 if no component is assigned to V_k.

There are two main types of constraints: *(1) structural*, which are related to the interaction between components, and *(2) hardware*, which are related to the components hardware requirements.

3.1 Structural Constraints

This type of constraints are of two types: *general* and *application-specific*. The *general* constraints are always considered in the deployment model and are related to basic allocation rules. The *application-specific* constraints are based on the case studies we considered (*Secure Web Container*, *Oryx2* and *Wordpress*).

We identified two main types of application-specific constraints regarding the components: interactions (conflict, collocation, exclusive deployment) and number of instances (require-provide, full deployment, deployment with a bounded number of instances)[4].

General Constraint. This case specifies that each component C_i must be allocated to at least one VM: $\sum_{i=1}^{N} a_{ik} \geq 1$, for all $k = \overline{1, M}$, except those being in *Exclusive Deployment* relation (see below).

Conflicts. This case corresponds to situations when there are conflictual components which cannot be deployed on the same VM. Considering that all conflicts between components are encoded in a matrix R (i.e. $R_{ij} = 1$ if C_i and C_j are conflictual components and $R_{ij} = 0$ otherwise), the constraints can be described as a set of linear inequalities: $a_{ik} + a_{jk} \leq 1$, $k = \overline{1, M}$, for all (i, j) such that $R_{ij} = 1$. Note that if the number of conflicts increases, the number of needed VMs also increases.

For example, for Wordpress application, Varnish component exhibits load balancing features. Hence, it should not be deployed on the same VM with HTTPLoadBalancer or DNSLoadBalancer. Moreover, Varnish and MySQL should not be deployed on the same VM because it is best practice to isolate the DBMS level of an application. Therefore, based on the notations in Figure 2, where each component has an assigned identifier, the coresponding constraints are:

$$a_{5k} + a_{ik} \leq 1, \quad i \in \{2, 3, 4\}, \quad k = \overline{1, M}$$

Collocation. This means that the components in the collocation relation should be deployed on the same VM. The collocation relation can be stored in a matrix D (i.e. $D_{ij} = 1$ if C_i and C_j should be collocated and $D_{ij} = 0$ otherwise) and the constraints can be described as a set of equalities: $a_{ik} = a_{jk}$, $k = \overline{1, M}$, for all (i, j) such that $D_{ij} = 1$.

For example, for Oryx2 application, components HDFS.DataNode and Spark.Worker must be deployed on the same VM. In this scenario, we also collocated Yarn.NodeManager because

[4]The component identifier occurring in the formulae in this section corresponds to the one from the figures in Section 2.

we used Yarn as a scheduler for Spark jobs:

$$a_{ik} = a_{9k}, \quad i \in \{5, 8\}, \quad k = \overline{1, M}$$

Exclusive deployment. There are cases when from a set of q components only one should be deployed in a deployment plan. Such a constraint can be described as:

$$H(\sum_{k=1}^{M} a_{i_1 k}) + H(\sum_{k=1}^{M} a_{i_2 k}) + ... + H(\sum_{k=1}^{M} a_{i_q k}) = 1,$$

where H is the Heaviside-like function defined as: $H(u) = 1$ if $u > 0$ and $H(u) = 0$ if $u = 0$.

For example, for Wordpress application, only one type of Balancer must be deployed (the Balancer components are HTTPLoadBalancer and DNSLoadBalancer). If HTTPLoadBalancer is deployed a caching component, in our case Varnish, should also be deployed leading to a different set of conflicts:

$$H(\sum_{k=1}^{M} a_{3k}) + H(\sum_{k=1}^{M} a_{4k}) = 1 \text{ and } H(\sum_{k=1}^{M} a_{3k}) + H(\sum_{k=1}^{M} a_{5k}) = 1$$

Require-Provide. A special case of interaction between components is when one component requires some functionalities offered by other components. Such an interaction induces constraints on the number of instances corresponding to the interacting components as follows: *(1)* C_i requires (consumes) at least n_{ij} instances of C_j and *(2)* C_j can serve (provides) at most m_{ij} instances of C_i. This can be written as:

$$n_{ij} \sum_{k=1}^{M} a_{ik} \leq m_{ij} \sum_{k=1}^{M} a_{jk}, \ n_{ij}, m_{ij} \in \mathbb{N}. \tag{1}$$

For example, for Wordpress application, the Wordpress component requires at least three instances of MySQL and MySQL can serve at most 2 Wordpress instances, leading to the constraint:

$$3 \sum_{k=1}^{M} a_{1k} \leq 2 \sum_{k=1}^{M} a_{2k}, \quad k = \overline{1, M}.$$

A related case is when for each set of n instances of component C_i a new instance of C_j should be deployed. This can be described as:

$$0 \leq n \sum_{k=1}^{M} a_{jk} - \sum_{k=1}^{M} a_{ik} < n, \ n \in \mathbb{N} \tag{2}$$

This constraint cannot be deduced from (1) because of the following. Taking in (1) $n_{ij} = 1$, we obtain an expression meaning that for m_{ij} instances of C_j one should have at least one instance of C_i (but there can be more). (2) is more specific requiring exactly one instance of C_j.

Full Deployment. There can be also cases when a component C_i must be deployed on all VMs (except on those which would induce conflicts on components). This can be expressed as:

$$\sum_{k=1}^{M} (a_{ik} + H(\sum_{j, R_{ij}=1} a_{jk})) = M$$

where R is the conflicts matrix and H is the Heaviside-like function defined above.

For example, for Oryx2 application, components HDFS.DataNode, YARN.NodeManager and Spark.Worker must be deployed on all VMs except those hosting conflicting components. Since $H(\sum_{j,R_{ij}=1} a_{jk}) = 0$, we have $\sum_{k=1}^{M} a_{ik} = M$, for $i \in \{5, 8, 9\}$.

Note that we do not allow in the application description the full deployment of two conflicting components. This is checked before invoking the recommendation engine.

Deployment with bounded number of instances. There are situations when the number of instances corresponding to a set of deployed components $\overline{C}$ should be equal, greater or less than some values. These types of constraints can be described as follows:

$$\sum_{i\in\overline{C}} \sum_{k=1}^{M} a_{ik} \ \langle \text{op} \rangle \ n, \quad \langle \text{op} \rangle \in \{=, \leq, \geq\}, \ n \in \mathbb{N}$$

For example, for Secure Web Container application, the total amount of instances of components Apache and Nginx must be at least 3 (level of redundancy):

$$\sum_{k=1}^{M} a_{2k} + \sum_{k=1}^{M} a_{3k} = 3, \quad k = \overline{1, M}$$

3.2 Constraints Related to Hardware Requirements and CPs Offers

These constraints specify the amount of resources required by the components assigned to a VM and assures that they do not overpass the VM characteristics as offered by the CPs. The set of available offers is obtained apriori by an Offer Management System component developed in the framework of our project and respects the minimum requirements of application components. So we start with a set of offers $\{Offer_1, Offer_2, \ldots, Offer_{ON}\}$, where ON is the number of possible CPs offers for VMs.

The challenging parts were to encode the CPs offers and to link them with the components hardware constraints. In order to overcome these issues, we introduced the following variables: *(1)* $vmType$ vector is used to identify the CPs offers ($vmType_k \in \{1, \ldots, ON\}$, $k = \overline{1, M}$); *(2)* vm^{HR_t} vectors are used to store a CPs offer ($vm_k^{HR_t} \in \mathbb{R}_+$, $k = \overline{1, M}$, $t = \overline{1, L}$). We consider in this paper three hardware resources: number of CPUs, memory, storage.

The next step was to link the $vmType$ with the allocation matrix a. Hence, we have:

$$\sum_{i=1}^{N} a_{ik} \geq 1 \wedge vmType_k = h \implies VP_k = Price^{Offer_h} \wedge vm_k^{HR_1} = HR_1^{Offer_h} \wedge \ldots \wedge vm_k^{HR_L} = HR_L^{Offer_h},$$

where $k = \overline{1, M}$ and $h = \overline{1, ON}$; $HR_t^{Offer_h}$ represents the CPUs number, memory size and storage size for offer h ($t = \overline{1, L}$). For example for the first offer ($h = 1$) corresponding values for ($Price^{Offer_1}$, $HR_1^{Offer_1}$, $HR_2^{Offer_1}$, $HR_3^{Offer_1}$) are (9.152$, 64, 488MB, 8GB). The formula above also specifies that the price of a VM contributes to the final price only if the machine is occupied. If the machine is not occupied, then it does not contribute to the final total leasing price. Hence, we have: $\sum_{i=1}^{N} a_{ik} = 0 \implies VP_k = 0, \quad k = \overline{1, M}$.

The last step was to ensure that the components requirements deployed on a VM do not exceed the resources of the selected offer:

$$\sum_{i=1}^{N} a_{ik} \cdot HR_t^i \leq vm_k^{HR_t}, \quad k = \overline{1, M}, \quad t = \overline{1, L}.$$

4 Encoding Approaches and Experimental Results

For solving the problem presented in Section 3, we used SMT solvers which exhibit optimization features, the so-called Optimization Modulo Theories (OMT) solvers. We did not have too many options in this regard:

1. OptiMathSAT [13] uses an inline architecture in which the SMT solver, MathSAT5 (`http://mathsat.fbk.eu`), is run only once and its internal SAT solver is modified to handle the search for the optima,
2. Symba [11] and νZ [4] both are based on an offline architecture in which the SMT solver Z3 [9] is incrementally called multiple times as a black-box.

We decided to perform a comparative analysis of the computation time between νZ and OptiMathSAT, since Symba is not maintained since 2014.

4.1 Variable Encoding

As the problem encoding, we decided to use linear arithmetic since the state-of-the-art OMT tools offer efficient decision procedures for this kind of theories. Based on the type of the variables from Section 3, we chose all variables (a, VP, $vmType$, vm_t^{HR}) to be of type `Real`[5].

The results are presented in Table 1. For solving the problem, we needed to know in advance the maximum number of VMs that the problem requires. This is not a fixed parameter in our approach, but is calculated using the structural constraints which involve number of instances (see `init#VMs` in the tables). This preprocessing step is not computationally expensive because it solves a constrained optimization problem with a number of variables equal to the number of components.

We studied the scalability of the OMT tools for the case studies from Section 2 from two perspectives: number of VMs offers, respectively number of deployed components. For *Secure Web Container* and *Oryx2* applications, we considered up to 500 CPs offers. Additionally, for the *Wordpress* application, we considered the number of instances of Wordpress component to be deployed. The set of offers was crawled from the Amazon CPs offers list.

All tests in this paper were done on an MacBook Pro with the following configuration: 3.1 GHz dual-core Intel Core i5, Turbo Boost up to 3.5GHz. In Table 1, we included only those results for which we obtained a result in 1 hour timeframe. One can observe that νZ does not scale well for high number of VMs offers and components, while OptiMathSAT even for small number of offers. From the logs, we observed that later reached many times the minimum but failed to prove it.

4.2 Symmetry Breaking

As one can see from Table 1, the naive application of general OMT solvers to our problem shows their lack of scalability. This is due to the large solutions space, which, however, might contain equivalent solutions. Therefore, in order to further reduce the search space, we used the following techniques:

1. a *heuristic* exploiting the particularities of the application, and
2. a *lex-leader method* introducing symmetry breaking constraints.

[5] A careful reader would notice the fact that the elements of a could be encoded as boolean. However, in order to express the terms from (3.2), the variables a, HR_t^i, and $vm_k^{HR_t}$ should be of compatible type. If a is considered to be boolean, one could use the ternary operator $If(a, 1., 0.)$, however this complicates more the SMT formulas.

Table 1: Scalability tests for νZ/OptiMathSAT tools. Time values are expressed in seconds.

Problem	**init# VMs**	**#offers = 20**		**#offers = 40**	**#offers = 100**	**#offers = 250**	**#offers = 500**
		νZ	Opti MathSAT	νZ	νZ	νZ	νZ
Oryx2	11	**6.62**	59.7	8.74	21.84	73.56	193.69
Secure Web Container	6	0.27	-	0.45	1.8	8.56	35.25
Wordpress min#inst = 4	10	16.39	-	120.84	1444.12	-	-
Wordpress min#inst = 5	12	775.7	-	-	-	-	-
Wordpress min#inst = 6	13	3516.19	-	-	-	-	-

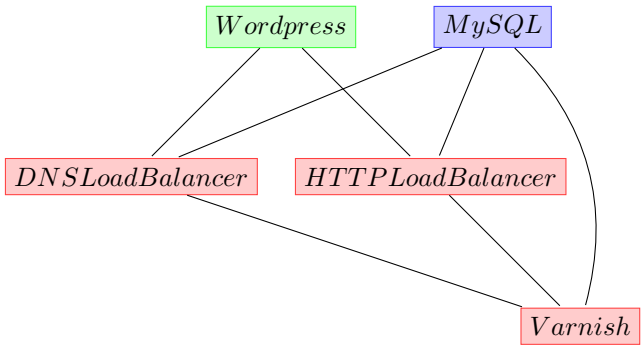

Figure 4: Wordpress min#inst=4 conflict graph. Red background components, being in *Exclusive Deployment* relation, are not taken into consideration for the construction of the cliques, so we have two cliques. From these two, $\overline{G} = \{Wordpress\}$ (deployment size is 4).

The constraints related to these techniques together with those presented in Section 3, are fed to the OMT tools.

Fixing Variable Values (FV). For any application, the components conflict graph can be constructed based on the restrictions of type *Conflicts*. In this graph it is possible to identify cliques (fully connected subgraphs), in which all pairs of components are conflictual so they cannot be placed on the same VM. Any component in a clique can have several deployed instances, therefore we can define the deployment size of a clique G as the total number of deployed instances of all components in G. In the following, we will denote by $\overline{G}$ the clique with maximal deployment size. Then each component of $\overline{G}$ is assigned to a different empty VM

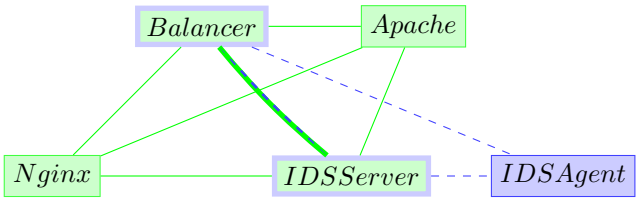

Figure 5: Secure Web Container conflict graph. The components with green background belong to the clique $\overline{G}$.

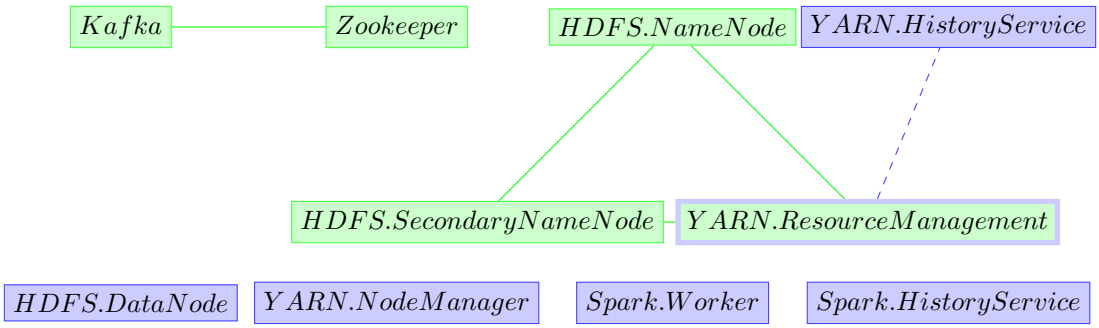

Figure 6: Oryx2 conflict graph. There are two cliques $\overline{G}$ which include the components with green background (the number of instances of Kafka is 2).

(by adding a constraint that fixes the variable a_{ik} to 1, where i is a component index from $\overline{G}$ and k is a VM index).

When the conflict graph is constructed, the components that are involved in restrictions of type *Exclusive deployments* are excluded. For example, in case of the Wordpress application, the DNSLoadBalancer, HTTPLoadBalancer and Varnish components are excluded (Figure 4) and in this case two cliques are identified: [Wordpress] and [MySQL]. In case of Secure Web Container (Figure 5) there are also two cliques: [Balancer, Apache, Nginx, IDSServer] and [Balancer, IDSServer, IDSAgent]. In both use cases the first clique plays the role of $\overline{G}$. The largest clique encountered in our use cases has 12 components instances and is found in case of Wordpress#12.

This approach introduces some additional constraints which fix some of the values of the assignment matrix variable reducing the number of candidate configurations.

To find all the cliques in the graph, we used an implementation of Bron-Kerbosch algorithm available in NetworkX library (`https://networkx.github.io`). Finding cliques inside a graph is a NP-hard problem, however in our case studies the graph size (given by the number of components) is not very large, so this preprocessing step does not increase significantly the execution time.

Lex-leader method. The motivation for using this kind of method came from the fact that in OMT techniques the search can revisit equivalent states over and over again. Hence it is desirable to eliminate as many equivalent (symmetric) states as possible. This is known as *symmetry breaking*: never explore two search states which are symmetric to each other since the result will be the same. One way to approach symmetry breaking is to add symmetry breaking constraints before search starts, hence making some symmetric solutions unacceptable while leaving at least one solution in each symmetric equivalence class.

We used lex-leader method for constructing symmetry-breaking ordering constraints for variable symmetries [8]. The idea is to predefine a canonical solution for each equivalence class of solutions. At this aim, we added constraints before invoking the OMT solver which are satisfied by the canonical solution and not by any others. We consider as canonical solutions those which fulfill an ordering of VMs based on either load (number of deployed instances per machine) or price:

1. **Virtual Machine Load (VML)**: $\sum_{i=1}^{N} a_{i1} \geq \sum_{i=1}^{N} a_{i2} \geq \ldots \geq \sum_{i=1}^{N} a_{iM}$
2. **Price (PR)**: $VP_1 \geq VP_2 \geq \ldots \geq VP_M$.

The comparative results of applying the strategies FV, VML and PR are presented in Table 2. By observing that FV and PR give the best timings, we combined them into PRFV strategy (Algorithm 1). In the case of PRFV, we used FV to fix, on separate VMs, the conflicting components and PR to sort the unassigned VMs. The list of VMs is not globally sorted but it is split in sublists which are sorted. This splitting is based on the structure of the clique with maximal deployment size ($\bar{G}$). More specifically, for each component in $\bar{G}$ the sublist containing the VMs on which its instances are deployed is decreasingly sorted based on price. Finally the VMs which do not contain instances of the components in $\bar{G}$ are decreasingly sorted (see Algorithm 1).

Algorithm 1 PRFV Algorithm

Find the clique $\overline{G}$
$k \leftarrow 1$ /*k - VM index*/
$CL = \{\}$ /*CL - constraints list*/
for each $C_j \in \overline{G}$ **do**
  $h \leftarrow k$
  for each C_j instances **do**
    $CL.add(a_{jk} = 1)$
    $k \leftarrow k + 1$
  end for
  $CL.add(VP_h \geq VP_{h+1} \geq \ldots \geq VP_{k-1})$
end for
$CL.add(VP_k \geq VP_{k+1} \geq \ldots \geq VP_M)$
return CL

The reported results consist in the average execution time and standard deviation from 10 independent runs. As expected, FV gives the best results for applications with high number of conflicts between components (*Secure Web Container* and *Oryx2*). Although VML and PR are similar approaches, PR outperforms VML in all cases. One can also observe that using them separately is not as efficient as combining them (in PRFV heuristic). From Table 2, even in some cases the timings of PR are better than those of PRFV, due to the high standard deviation of the first one, one cannot assume that PR always outperforms PRFV.

Also OptiMathSAT scales better by using symmetry breaking but the timings are worse than those of νZ.

5 Conclusions

In this paper, we analyzed the scalability of state-of-the-art OMT tools for optimization problems coming from Cloud resource management. The outcome was that general OMT tools do not scale well for problems involving a large number of disjunctive constraints. Our improvement solution was to use graph theory (clique detection) and constraint programming (symmetry breaking) approaches before invoking the OMT tools in order to reduce the search space.

As future work, by observing that the cost function in the optimization problem is a sum of variables whose values are a deterministic consequence of the current truth assignment, for better computational results, we plan to encode the problem as a MaxSMT one by means of the assert-soft constructs. We also plan to encode our problems as a integer linear program and use appropriate solvers to compare them with the methods introduced in this paper.

Table 2: Scalability analysis: for νZ, time to find an optimal solution (sec) in case of applying different symmetry breaking strategies. FV – fixing variables values, VML – order on VM load (# of deployed instances), PR – order on VM price, PRFV – combination of PR and FV.

Problem	**Strategy**	**#offers=20**	**#offers=40**	**#offers=100**	**#offers=250**	**#offers=500**
Oryx2 **init#VMs =11**	FV	0.45(±0.12)	0.51(±0.06)	2.28(±0.22)	6.13(±0.72)	114.26(±14.90)
	VML	1.00(±0.10)	1.60(±0.14)	3.06(±0.44)	10.07(±1.23)	208.33(±25.47)
	PR	0.55(±0.15)	0.68(±0.08)	**1.11(±0.19)**	5.61(±1.10)	**113.85(±26.36)**
	PRFV	**0.20(±0.03)**	**0.48(±0.07)**	2.41(±0.40)	**4.31(±0.53)**	176.08(±25.75)
Secure Web Container **init#VMs =6**	FV	**0.09(±0.01)**	0.10(±0.02)	0.41(±0.07)	9.96(±1.03)	56.95(±5.70)
	VML	0.19(±0.01)	0.39(±0.08)	1.64(±0.23)	16.93(±1.44)	112.68(±9.03)
	PR	0.19(±0.02)	0.32(±0.04)	1.42(±0.23)	11.54(±1.93)	76.69(±9.80)
	PRFV	**0.09(±0.02)**	**0.08(±0.01)**	**0.30(±0.05)**	**7.45(±0.82)**	**43.00(±4.51)**
Wordpress min#inst=4 **init#VMs =10**	FV	0.76(±0.14)	1.74(±0.27)	6.69(±0.88)	43.74(±4.03)	229.76(±16.37)
	VML	0.99(±0.11)	1.66(±0.16)	5.76(±0.58)	37.90(±5.40)	191.84(±16.68)
	PR	0.48(±0.09)	1.05(±0.10)	3.42(±0.36)	23.51(±4.36)	129.86(±22.14)
	PRFV	**0.31(±0.16)**	**0.62(±0.07)**	**2.55(±0.17)**	**21.82(±2.03)**	**128.47(±10.56)**
Wordpress min#inst=5 **init#VMs =12**	FV	1.73(±0.25)	36.79(±3.81)	28.59(±7.08)	1175.08(±283.0)	752.03(±107.26)
	VML	1.57(±0.13)	3.00(±0.42)	9.54(±1.10)	59.13(±2.86)	289.02(±30.47)
	PR	0.92(±0.18)	2.48(±0.64)	5.22(±0.77)	40.85(±0.00)	**183.22(±38.85)**
	PRFV	**0.30(±0.05)**	**0.63(±0.19)**	**3.53(±0.47)**	**9.89(±0.63)**	**183.38(±19.32)**
Wordpress min#inst=6 **init#VMs =13**	FV	1.82(±0.22)	7.59(±1.26)	1345.17(±438.3)	2943.98(±793.6)	1230.36(±309.20)
	VML	1.76(±0.19)	2.62(±0.22)	19.68(±2.00)	70.00(±9.90)	323.08(±23.88)
	PR	0.82(±0.09)	1.93(±0.31)	4.02(±1.28)	15.69(±1.54)	**202.36(±24.15)**
	PRFV	**0.35(±0.05)**	**0.78(±0.08)**	**2.08(±0.16)**	**10.47(±1.23)**	210.30(±17.83)

Table 3: Scalability analysis: for νZ, time to find an optimal solution (sec) in case of applying different symmetry breaking strategies for Wordpress application with high number of components instances.

Problem	**init# VMs**	**Strategy**	**#offers =20**	**#offers =40**	**#offers =100**	**#offers =250**	**#offers =500**
Wordpress min#inst=7	15	PR	1.1(±0.1)	2.4(±0.2)	6.8(±0.6)	46.7(±7.5)	228.6(±44.0)
		PRFV	**0.5(±0.08)**	**0.8(±0.07)**	**2.69(±0.2)**	**43.5(±4.4)**	**70.6(±5.7)**
Wordpress min#inst=8	17	PR	1.3(±0.3)	3.3(±0.4)	10.6(±3.7)	65.3(±36.5)	334.2(±53.8)
		PRFV	**0.6(±0.07)**	**1.3(±0.1)**	**3.82(±0.3)**	**17.47(±0.9)**	**318.3(±36.9)**
Wordpress min#inst=9	18	PR	1.6(±0.3)	3.8(±0.6)	10.2(±1.0)	64.9(±7.9)	368.4(±54.1)
		PRFV	**0.7(±0.08)**	**1.0(±0.1)**	**7.4(±0.5)**	**18.8(±1.8)**	**97.6(±10.1)**
Wordpress min#inst=10	20	PR	2.0(±0.2)	5.0(±1.0)	12.5(±1.1)	89.1(±8.9)	**387.2(±70.3)**
		PRFV	**0.8(±0.07)**	**1.6(±0.1)**	**8.4(±0.5)**	**70.8(±7.3)**	392.3(±29.3)
Wordpress min#inst=11	22	PR	2.6(±0.7)	5.7(±0.6)	14.4(±1.2)	115.5(±15.7)	545.4(±76.1)
		PRFV	**1.0(±0.1)**	**1.6(±0.2)**	**10.1(±0.6)**	**29.2(±3.6)**	**133.0(±13.0)**
Wordpress min#inst=12	23	PR	2.7(±0.3)	6.5(±1.0)	286.1(±87.1)	1000 (±205.1)	585.4(±70.7)
		PRFV	**0.9(±0.09)**	**2.1(±0.1)**	**7.0(±0.6)**	**33.8(±4.2)**	**503.9(±24.5)**

Acknowledgements. We thank Erika Ábrahám for for many helpful discussions on SMT, Patrick Trentin and Roberto Sebastiani for their support with OptiMathSAT.

References

[1] E. Ábrahám, F. Corzilius, E. B. Johnsen, G. Kremer, and J. Mauro. Zephyrus2: On the fly deployment optimization using SMT and CP technologies. In *Dependable Software Engineering: Theories, Tools, and Applications - Second International Symposium, SETTA 2016, Beijing, China, November 9-11, 2016, Proceedings*, pages 229–245, 2016.

[2] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu. SAT modulo monotonic theories. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3702–3709, 2015.

[3] S. Bayless, N. Kodirov, I. Beschastnikh, H. H. Hoos, and A. J. Hu. Scalable constraint-based virtual data center allocation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 546–554, 2017.

[4] N. Bjørner, A. Phan, and L. Fleckenstein. νZ - an optimizing SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 194–199, 2015.

[5] V. Casola, A. D. Benedictis, M. Erascu, J. Modic, and M. Rak. Automatically enforcing security slas in the cloud. *IEEE Trans. Services Computing*, 10(5):741–755, 2017.

[6] C. Chen, S. Yan, G. Zhao, B. Lee, and S. Singhal. A systematic framework enabling automatic conflict detection and explanation in cloud service selection for enterprises. In *2012 IEEE Fifth International Conference on Cloud Computing, Honolulu, HI, USA, June 24-29, 2012*, pages 883–890, 2012.

[7] R. D. Cosmo, M. Lienhardt, R. Treinen, S. Zacchiroli, J. Zwolakowski, A. Eiche, and A. Agahi. Automated synthesis and deployment of cloud applications. In *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, pages 211–222, 2014.

[8] J. M. Crawford, M. L. Ginsberg, E. M. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, pages 148–159, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[9] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[10] M. Guzek, P. Bouvry, and E. Talbi. A survey of evolutionary computation for resource management of processing in cloud computing [review article]. *IEEE Comp. Int. Mag.*, 10(2):53–67, 2015.

[11] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik. Symbolic optimization with SMT solvers. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 607–618, 2014.

[12] F. Micota, M. Eraşcu, and D. Zaharie. Constraint satisfaction approaches in cloud resource selection for component based applications. 2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing. to appear.

[13] R. Sebastiani and P. Trentin. OptiMathSAT: A tool for optimization modulo theories. In D. Kroening and C. S. Păsăreanu, editors, *Computer Aided Verification*, pages 447–454, Cham, 2015. Springer International Publishing.