Symmetry Breaking for the Cloud Resource Allocation Problem

Francesca Drăguț[1,a] and Adelina Anescu[1,b]

[1] *Faculty of Mathematics and Informatics, West University of Timișoara*
[a] francesca.dragut01@e-uvt.ro
[b] adelina.anescu01@e-uvt.ro

Abstract

In this paper, we study the problem of Cloud Resource Allocation on the concrete example of Secure Web Container. Our main focus is on symmetry breaking techniques and how they impact the performance of finding the optimal solution of deploying a component-based application in Cloud. We study three symmetry breaking techniques on the Secure Web Container example: lexicographic ordering (LX) on columns, fixed values (FV) and price-based ordering (PR) in ascending and descending order. We reach the conclusion that while FV is the fastest symmetry breaker on our example, it may not be feasible on applications that have few component-conflicts. Considering implementation difficulty, reliability and execution time, we reach the conclusion that PR in descending order is the most feasible symmetry breaker for the studied example.

1 Introduction

The problem of Cloud Resource Allocation refers to deploying component-based applications on virtual machines (with price, CPU, storage, memory specifications), at the lowest cost possible. This is an NP-hard, constraint satisfaction problem. In order to optimize the search time, we can use symmetry breaking techniques, which exclude the already visited parts in the search tree, as well as the solutions which are symmetric to the visited parts.

To solve this problem, we relate to existing research (see [1]). This is a comprehensive study of symmetry breaking in Cloud Deployment, which treats real-life examples. Out of the examples presented in this research, we study Secure Web Container (for more details, see [2]).

Our aim in this study is to describe and develop symmetry breaking methods for this problem on the example of Secure Web Container, discussed in [1]. Moreover, we want to identify the most feasible symmetry breaking technique for this example, considering parameters such as implementation complexity, stability and running time.

2 Theoretical Analysis

Our study has a wide theoretical basis, which we will present in this section. The main concepts that we will operate with include constraint satisfaction problems, optimization of search algorithms, symmetry breaking techniques and how they all connect.

First of all, the problem that we study is a constraint satisfaction problem (CSP). A CSP operates with a set of variables and a set of constraints, to which the variables and the relations between the variables are subjected to. After solving a constraint satisfaction problem, we have two possible answers - *satisfiable* or *unsatisfiable*. If the problem is satisfiable, it means that the variables subjected to constraints have a corresponding set of values (which are formalized into an assignment matrix), which satisfy the problem and that we can obtain a model for the solution.

A constraint satisfaction problem takes into consideration more possible arrangements of the variables into the assignment matrix. The assignment matrix can be large and it can have more possible satisfiable arrangements of the variables. Therefore, we have a large search space for the optimal solution, which makes the problem NP-hard. This is why we need to optimize the process, by adding symmetry breaking techniques.

The need of optimization means that the Cloud Resource Allocation problem is also a constraint optimization problem (COP). We solve the CSP part by checking the satisfiability of the encoding and we solve the COP part by enforcing the total price to be minimal.

Before defining the symmetry breaking techniques, we need to define what symmetries mean for our problem. The assignment matrix formalizes the distribution of the components on virtual machines. In this matrix, the columns represent the virtual machines, while the each row represents a component of the application about to be deployed. Each cell of the matrix represents an assignemnt variable, which is either 1 or 0, depending on whether the component is placed or not on the corresponding virtual machine.

In the end, we have a set of assignment matrices, which all represent satisfiable solutions for the problem. Out of this set, we need to find the optimal one. The problem is that in this set, many matrices are symmetric, meaning that even though they are not the same, they lead to the same solution. This prolongs the searching process, because the search will consider the same matrix, in different forms, more times than necessary. One step to overcome this issue is to apply symmetry breaking techniques, which will prevent the search to iterate through already visited solutions.

The symmetry breaking techniques are of three types [3]:

- **Reformulation**. Implies redesigning of the problem; thus, there is no mathematical formalization of this technique for general problems.
- **Static symmetry breaking**. Implies adding symmetry breaking constraints before the actual search, to eliminate the non-compliant solutions in the initial stage.
- **Dynamic symmetry breaking**. Implies deleting symmetries during search,

while also adapting the search program.

In our study we use static symmetry breaking, because it is efficient, eliminating symmetries in the beginning. Static symmetry breaking also has solid mathematical formalization, unlike reformulation, making it adaptable to any problem. Unlike the dynamic symmetry breaking technique, this technique is fairly simple to implement, because it uses exact algorithms.

The static symmetry breaking technique operates with symmetry breaking constraints. These symmetry breaking constraints are of more types, but in this study we included the following:

- **Lexicographic ordering**
  This method adds constraints that enforce two adjacent columns to be in decreasing lexicographic order.
- **Price-based ordering**
  In the problem formalization, each virtual machine has a t number of types. Each type corresponds to a set of values for the virtual machine (price, CPU, memory, storage). When we apply *price-based ordering*, we sort the vector of types decreasingly by price.
- **Fixed values** To reduce the search space, we can start with some set values. These values are fixed based on the component-conflicts, which are application-specific constraints. The *fixed values* technique requires some prior deductive reasoning.

We apply all of these concepts on a concrete example. Our study is highly based on the work presented in [1], which was a reference point in our work. In this paper, there are several examples considered, on which the authors apply symmetry breaking techniques. Out of the examples, we considered the Secure Web Container, which we will present and formalize in the following sections.

Secure Web Container Example[1]

This is a real-life example of a component-based application, which needs to be deployed in Cloud. The Secure Web Container has 5 components, which, in this example, need to be placed on 6 virtual machines, each having 20 possible types, which leads us to 120 VM offers (example was taken from [4]).

The application specific constraints are the following:

- C1 cannot be placed on the same VM as any other component;
- C4 cannot be placed on the same VM as any other component;
- C2 cannot be placed on the same VM as C3;
- C1 must be placed on exactly 1 VM;
- The total number of assignments for C2 and C3 must be at least 3;
- C5 must be placed on each machine, except for the ones that already have either C1 or C4 deployed on them;
- For each 10 deployments of C5 on a VM, there must be at least 1 deployment of C4.

3 Experimental Design

We implemented the example presented in Section 2 in the Z3 environment. Z3 is a Satisfiability Modulo Theory (SMT) solver. SMT solvers are used when working with quantified formulas [5].

We used the online version [6] of the Z3 software, as it had enough capabilities to support our tests.

We formalized the code that we used and transformed it into mathematical formulas, which can be adapted to any SMT solver (see Equations 1-19 in Section 3.1).

3.1 Formalization of the Cloud Resource Allocation Problem

In Z3, there are three steps in the solving process: **declaration** of the variables, **assertion** of the constraints and the actual **solving** of the problem, which minimizes the total price and checks for satisfiability (if the result is satisfiable, a model can be shown on request).

DECLARE:

- Declare variables for each element of the assignment matrix:

$$Ci\_VMj, \ \ \forall i = \overline{1,n}, \ \ j = \overline{1,m} \tag{1}$$

where n =number of components, m =number of virtual machines

- Declare variables for CPU, memory and storage for each component:

$$\begin{array}{l} cpu\_Ci, \ \ \forall i = \overline{1,n} \\ mem\_Ci, \ \ \forall i = \overline{1,n} \\ storage\_Ci, \ \ \forall i = \overline{1,n} \end{array} \tag{2}$$

- Declare VM types:

$$VMjType, \ \ \forall j = \overline{1,m} \tag{3}$$

- Declare variables for CPU, memory and storage for each VM:

$$\begin{array}{l} cpu\_VMj, \ \ \forall j = \overline{1,m} \\ mem\_VMj, \ \ \forall j = \overline{1,m} \\ storage\_VMj, \ \ \forall j = \overline{1,m} \end{array} \tag{4}$$

ASSERT:

- Matrix elements should be either 0 or 1:

$$Ci\_VMj = 0 \ \ \vee \ \ Ci\_VMj = 1, \ \ \forall i = \overline{1,n}, \ \forall j = \overline{1,m} \tag{5}$$

- Set CPU, memory and storage for each component with desired values
- If there are components on a VM, its type is non-0:

$$(Ci\_VMj = 1) \Rightarrow VMj\_Type \neq 0, \ \ \forall i = \overline{1,n}, \ \forall j = \overline{1,m} \tag{6}$$

- The price for each VM should be non-negative:

$$price\_VMj \geq 0, \ \ \forall j = \overline{1,m} \tag{7}$$

- If the VM has no components, its price will be set to 0:

$$\sum_{i=1}^{m} Ci\_VMj = 0 \Rightarrow price\_VMj = 0, \ \ \forall j = \overline{1,m} \tag{8}$$

- For each VM, for each type, set price, CPU, memory and storage values:

$$\sum_{i=1}^{n} Ci\_VMj \geq 1 \ \ \wedge \ \ VMj\_Type = k \Rightarrow \begin{cases} price\_VMj = value \\ CPU\_VMj = value \\ mem\_VMj = value \\ storage\_VMj = value \end{cases} \tag{9}$$

- General constraints - needed for any application:

  Basic Allocation:

  This constraint enforces all of the components to be placed on at least one virtual machine. This way, we ensure that all of the components of the application are deployed in Cloud.

$$\sum_{j=1}^{m} Ci\_VMj \geq 1, \ \ \forall i = \overline{1,n} \tag{10}$$

  Capacity:

  The capacity constraints make sure that the total memory, storage and CPU of each component does not overpass the capabilities of the VM it was deployed on. For example, if we have 3 instances of C1 on VM1, where the CPU required by an instance of C1 is 4GB, leading to a total of 12GB and the capabilities of VM1 are 8GB of CPU, the instances of C1 will not fit in on VM1, because they would violate the capacity constraints.

$$\sum_{i=1}^{n} (Ci\_VMj \cdot Ci\_mem) \leq mem\_VMj, \ \ \forall j = \overline{1,m} \tag{11}$$

$$\sum_{i=1}^{n} (Ci\_VMj \cdot Ci\_storage) \leq storage\_VMj, \ \ \forall j = \overline{1,m} \tag{12}$$

$$\sum_{i=1}^{n} (Ci\_VMj \cdot Ci\_CPU) \leq CPU\_VMj, \ \ \forall j = \overline{1,m} \tag{13}$$

- Application specific constraints for the Secure Web Container example:

  Component conflicts:

$$C1\_VMj + C2\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C1\_VMj + C3\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C1\_VMj + C4\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C1\_VMj + C5\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C2\_VMj + C3\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C4\_VMj + C2\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C4\_VMj + C3\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

$$C4\_VMj + C5\_VMj \leq 1, \ \ \forall j = \overline{1,m}$$

(14)

Other constraints:

$$\sum_{j=1}^{m} C1\_VMj = 1 \tag{15}$$

$$\sum_{j=1}^{m} (C2\_VMj + C3\_VMj) \geq 3 \tag{16}$$

$$C1\_VMj + C4\_VMj + C5\_VMj = \begin{cases} 1, & if \sum_{i=1}^{n} Ci\_VMj \geq 1, \ \ \forall j = \overline{1,m} \\ 0, & otherwise \end{cases} \tag{17}$$

$$(10 \cdot \sum_{j=1}^{m} C4\_VMj) - (\sum_{j=1}^{m} C5\_VMj) \in (0, 10] \tag{18}$$

SOLVE:
Minimize the sum of the price (see Equation 19). After minimizing, check for satisfiability. If the result is "sat", one can also request a model for the solution, which will output a satisfiable assignment for all the declared variables.

$$minimize(\sum_{j=1}^{m} price\_VMj) \tag{19}$$

3.2 Adding Symmetry Breaking Techiques

3.2.1 Lexicographic Ordering

Row/column symmetry breaking with lexicographic ordering can be viewed as sorting numbers in base 2, which are represented by the row/column vectors in the assignment matrix. With lexicographic ordering, the elements in the assignment matrix are constrained to be placed in lexicographic order. Symmetry breaking can be done on rows, columns or both rows and columns at the same time.

Lexicographically ordering rows or columns separately breaks all symmetries, while ordering both of them at the same time leaves unbroken symmetries [7]. Since the placement of components in the assignment matrix cannot be modified and rows are interdependent with columns, lexicographically ordering both rows and columns at the same time will not break all the symmetries.

By ordering the matrices lexicographically, we end up with only one matrix from a symmetry group. This method orders the matrices and therefore, the symmetric matrices will become undistinguishable. Thus, symmetric matrices will not be revisited.

Having established that lexicographical ordering can be done in three ways, we have to analyze each method:

- *Rows.* Lexicographically ordering the rows of the assignment matrix can be done, but it was not studied in this paper;
- *Columns.* Lexicographically ordering the columns of the assignment matrix led us to a satisfiable result, with minimum price maintained, while offering the output faster than without symmetry breaking;
- *Both rows and columns.* Lexicographical ordering of both rows and columns was not implemented in this experimental design, due to encoding complexity.

Given the information presented above, we focused on ordering the assignment matrix lexicographically on columns.

In equations 20, 21 and 22, we present the mathematical formalization of the code we used for lexicographical ordering on columns in the Secure Web Container example. The implementation in Z3 can be seen in detail at [8].

$$C1\_VMj \geq C1\_VM(j+1), \;\; \forall j = \overline{1,m} \tag{20}$$

$$(C1\_VMj = C1\_VM(j+1)) \Rightarrow (C2\_VMj \geq C2\_VM(j+1)), \;\; \forall j = \overline{1,m} \tag{21}$$

$$(C1\_VMj = C1\_VM(j{+}1) \wedge C2\_VMj = C2\_VM(j{+}1)) \Rightarrow (C3\_VMj \geq C3\_VM(j{+}1)) \tag{22}$$

All of the remaining asserts follow the model from Equation 22. We continue adding asserts for the cases when more components have the same value on two adjacent

virtual machines. We increase the index of the virtual machine and repeat this procedure for every pair of adjacent virtual machines.

3.2.2 Fixed Values

Using the fixed values technique, certain values in the assignment matrix are set beforehand. However, these values are not randomly chosen. They are fixed considering the application-specific constraints. For the Secure Web Container, the values in Table 1 were fixed before the search.

We know that C1 is in conflict with all of the other components. We also know that C1 must be on exactly one VM. Therefore, we can start by placing C1 on VM1. By placing C1 on VM1, we already know the whole distribution of components on VM1 - 1 for C1 and 0 for all of the other components. Also, since C1 must be on exactly one VM, the row corresponding to C1 will be 0 for all of the other VMs, except for VM1.

We also know that C4 is in conflict with all of the other components. This means that the VMs that it will be placed on must be empty. We will place C4 on VM2, leaving the assignemnts for all of the other components on VM2 be 0. We do not have any further clue about the number of assignments for C4, so we will only assign it on VM2 for now.

Because of the component conflicts for C1 and C4, VM1 and VM4 are now unavailable. We know that C2 and C3 cannot be placed on the same VM, as they are in conflict. Nevertheless, they must be assigned at least 3 times in total. We will assign a C2 on VM3, which is the next available VM. For VM3, we also know that the assignments for C1, C3 and C4 must be 0, because C2 is in conflict with all of them. We now need to place a C3, which we will place on VM4. Because C3 is in conflict with C1, C2 and C4, the assignments of these components on VM3 will be 0.

So far, these are the values we can fix, without making risky assumptions. We have the minimum of assignments which will not alter the final output or the minimum price. After fixing these values, based on the component-based conflicts, we only have 10 out of 30 assignments to make, which diminishes the search space significantly.

	VM1	VM2	VM3	VM4	VM5	VM6
C1	1	0	0	0	0	0
C2	0	0	1	0		
C3	0	0	0	1		
C4	0	1	0	0		
C5	0	0				

Table 1: The fixed values in the matrix for the Secure Web Container example, before the search starts.

FV is a row-symmetry breaker, that is highly dependent on the number of component-conflicts in the application. This means that the execution time varies depending on the number of these conflicts.

We tried to prove this, by eliminating some component-conflicts from the FV code.

We eliminated all component-conflicts, except for the ones for C1. Thus, we have the conflicts for C1 with all of the other components, the constraint that C1 must be on exactly one VM and the other application-specific constraints, that do not influence the fixing of the values. Since we only have the conflicts for C1, we can only fix the column for VM1 (where C1 has value 1 and the other components have value 0) and the row for C1, where the only assignment of C1 is on VM1.

In the case presented above, we have much fewer fixed values (only 10). Because the constraints change, the minimum price also changes, but this is not the focus of the experiment, because the size of the problem is the same, so the execution times are comparable (see Section 4 for more details).

3.2.3 Price-Based Ordering

For the *price-based ordering* technique, the prices of the virtual machines are set to be in some kind of order.

NO PR	VM1	VM2	VM3	VM4	VM5	VM6
C1	1	0	0	0	0	0
C2	0	1	0	0	1	0
C3	0	0	0	0	0	1
C4	0	0	0	1	0	0
C5	0	1	0	0	1	1
price	379	402	0	1288	402	1288
CPU	4	4	4	8	4	8
mem	30500	15000	30500	68400	15000	68400
storage	1000	2000	1000	2000	2000	2000
type	15	13	15	12	13	12

Table 2: Model of the encoding for Secure Web Container without applying PR.

PR desc	VM1	VM2	VM3	VM4	VM5	VM6
C1	0	0	0	0	1	0
C2	0	0	1	1	0	0
C3	1	0	0	0	0	0
C4	0	1	0	0	0	0
C5	1	0	1	1	0	0
price	1288	1288	402	402	379	0
CPU	8	8	4	4	4	3
mem	68400	68400	15000	15000	30500	1700
storage	2000	2000	2000	2000	1000	1000
type	12	12	13	13	15	2

Table 3:
Model of the encoding for Secure Web Container with PR (when VMs are sorted in descending order.

In Equation 23 we can see the formalization for the descending order and in Equation 24, we can see the formalization for the ascending order.

$$price\_VM1 \geq price\_VM2 \geq ... \geq price\_VM_m \tag{23}$$

$$price\_VM1 \leq price\_VM2 \leq ... \leq price\_VM_m \tag{24}$$

In order to help us present how the price-based ordering technique operates, we built Tables 2 and 3. The columns in Table 2 match the columns in Table 3 by color. When ordering by price, there is a swap done between virtual machines, considering price. Moreover, there is some renaming that takes place. When we move VM1 on position 5, after applying PR, the name of VM1 also changes to match the index. Therefore, in the end, we have the assignment matrix ordered by price, with index naming being maintained.

When prices are ordered, the search time reduces, because the search will be linear. This symmetry breaker eliminates the backtracking that would occur in order to check if there is any price that is more favourable. One disadvantage of this method occurs when more virtual machines have the same price, because not all symmetries will be broken.

4 Results

In this section, we compare the results that we obtained working on the Secure Web Container example. We analyze the time of finding the optimal solution without and with symmetry breaking techniques and we present a comparison between these, in order to outline their importance. We also aim to find the best symmetry breaking technique, by considering three parameters: execution time, stability and reliability.

Symm. Breakers	Time (s)
No Symm. Br.	0.779
PR (desc.)	0.390
PR (asc.)	0.424
LX	0.470
FV (all constraints)	0.352
FV (elim. constr.)	0.458

Table 4:
Comparison of the execution times for the Secure Web Container example, without symmetry breaking techniques, with price-based ordering (PR), lexicographical ordering (LX), fixed values (FV) and their variants.

As we can see in Table 4, there is a significant difference between using symmetry breaking techniques and not using them. For all of the symmetry breaking techniques, the time is almost half of the original one. We will analyze each technique, considering the parameters discussed above.

Considering execution time, FV stands out to be the best. But, as we have seen,

FV's performance is relative to the number of component-conflicts in the application. Also, combining FV with other symmetry breaking techniques, execution time may increase, but this is, again, relative to the application that we study.

Slightly slower, but more stable is the descending ordering by price. This method delivered the results fast, while also being more independent to the application that it is applied on. Also, the ascending ordering by price is relatively fast, as it comes right after descending PR in the execution time ranking.

The next method, considering execution time, is FV with some eliminated component-conflicts. Even though it is relatively fast compared to no symmetry breaking, the difference to the FV method with all constraints is significant.

The last in the ranking is LX, which performed most poorly out of the studied symmetry breakers. Applying LX still reduced the time considerably, but the other methods turned out to be faster. The lexicographic ordering that we applied was column-wise, as studies suggested that this approach is the most effective [1], [7]. Nevertheless, the implementation is lengthy and the execution time is not the fastest.

We learned that the most feasible symmetry breaking technique for the Secure Web Container example was descending PR. The implementation of this method is undemanding, while providing fast execution time for the optimal solution. Also, it is a stable method, meaning that it is not influenced as much by external factors, such as the application-specific constraints.

5 Conclusions and Future Work

In this paper, we studied the Cloud Resource Allocation Problem, with the main focus being on symmetry breaking techniques. The Cloud Resource Allocation problem is wide and complex, so we focused on a case study, namely Secure Web Container. We wanted to see how symmetry breakers improve the running time on this example and which one is the most feasible technique.

The symmetry breakers that we considered in our study are the following: Lexicographical Ordering (LX), Price-Based Ordering (PR) and Fixed Values (FV).

The most important results are related to the feasibility of the descending PR technique. Even though the running time was average, it turned out to be the most stable one, being able to produce fast and reliable results, regardless of the input. On the other hand, FV turned out to highly depend on the number of component-conflicts, which are specific to each application and can vary. We learned that the smaller the number of conflicts, the less efficient FV is. Lexicographical ordering provided the slowest results. Moreover, the implementation is lengthy and does not guarantee reliable results (eg. the row-wise ordering provided an unsatisfiable result).

Even though we reached good results, there are still points to improve in our study. The most important thing to be improved would be to find a more precise and reliable algorithm for row-wise lexicographic ordering.

This study was introductory, so the number of considered symmetry breakers was

rather low. In the future, we plan to add more symmetry breakers in the study, in order to see how they behave and to see if we can find an even better one.

Acknowledgements

We would like to mention [1], for providing the base of our study, with the theoretical and experimental aspects.

We would also like to mention [9] for the detailed help in structuring and writing this paper.

References

[1] Madalina Erascu, Flavia Micota, and Daniela Zaharie. Scalable optimal deployment in the cloud of component-based applications using optimization modulo theory, mathematical programming and symmetry breaking. *Journal of Logical and Algebraic Methods in Programming, volume 121*, 2021.

[2] Valentina Casola, Alessandra De Benedictis, Massimiliano Rak, Jolanda Modic, and Madalina Erascu. Automatically enforcing security slas in the cloud. *IEEE Transactions on Services Computing 10*, pages 741–755, 2016.

[3] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming. Chapter 10. Symmetry in Constraint Programming.* Elsevier Science Inc., USA, 2006.

[4] https://github.com/Maneuver-PED/RecommendationEngine/blob/master/journal/Encoding_AllCombinationsOffers/noSymBreaking/output_Z3_SolverInt_SB_Enc_AllCombinationsOffers/SMT2/SecureWebContainer-offers_20.smt2.

[5] Andrew Reynolds and Viktor Kuncak. Induction for SMT Solvers. pages 80–98, 2015.

[6] Z3 Online Demonstrator. https://compsys-tools.ens-lyon.fr/z3/. Accessed: 2022-01-01.

[7] Pascal Van Hentenryck. *Principles and Practice of Constraint Programming - CP 2002: 8th International Conference, CP 2002 Ithaca, NY, USA, September 9–13, 2002 Proceedings.* 2002.

[8] https://github.com/francescadragut/Symmetry-Breaking-for-the-Cloud-Resource-Allocation-Problem/blob/main/FV_DeletedConflicts.smt2.

[9] Tomislav Hengl and Michael Gould. Rules of thumb for writing research articles. 01 2002.