# An Efficient Communication Aware Heuristic for Multiple Cloud Application Placement

Pedro Silva, Christian Pérez

# An Efficient Communication Aware Heuristic for Multiple Cloud Application Placement

Pedro Silva* and Christian Perez*
pedro.silva@inria.fr, christian.perez@inria.fr

* Univ. Lyon, Inria, CNRS, ENS de Lyon, UCBL 1, LIP

**Abstract.** To deploy a distributed application on the cloud, cost, resource and communication constraints have to be considered to select the most suitable Virtual Machines (VMs), from private and public cloud providers. This process becomes very complex in large scale scenarios and, as this problem is NP-Hard, its automation must take scalability into consideration. In this work, we propose a heuristic able to calculate initial placements for distributed component-based applications on possibly multiple clouds with the objective of minimizing VM *renting costs* while satisfying applications' *resource* and *communication* constraints. We evaluate the heuristic performance and determine its limitations by comparing it to other placement approaches, namely exact algorithms and meta-heuristics. We show that the proposed heuristic is able to compute a good solution much faster than them.

## 1 Introduction

To *place* an application onto the Cloud, in the context of Infrastructure as a Service (IaaS), a designer must choose the best set of machines, generally virtual machines, from public and private *cloud providers*, which satisfies application performance constraints. When the placement aims at minimizing renting costs, the abundant number of available cloud providers and their offerings makes this task challenging. Although automation becomes crucial, the placement problem is *NP-hard*, and hence *scalability* must be taken in consideration, particularly in the cases where applications are large and *time constraints* are tight.

In spite of important contributions made by previous works, issues concerning, mainly, *scalability* and the *modeling of communication constraints* are still open, in particular in the case of multiple cloud deployment. Scalability issues are observed in works that propose time consuming solutions based on exact algorithms, meta-heuristics or solvers. Issues related to modeling communication constraints become apparent in works that either do not consider them at all or assume complete knowledge of cloud network and application topologies. In reality, users usually do not have access to the exact network topology of cloud provider data centers. Furthermore, due to hardware virtualization, the identity and location of physical machines where VM instances run may vary. Hence, placement algorithms that rely on those assumptions may not work correctly.

We tackle the problem of finding an *initial* placement for *distributed applications* modeled as *component-based* applications on *multiple clouds*. For brevity, we call this problem *CAPDAMP*, for *Communication Aware Placement of Distributed Applications on the Multi-cloud Problem*. The objective is to map each *application component* to an instance of a *virtual machine* (VM) minimizing renting *costs* and *satisfying* resource and communication constraints. Components can be any piece of code. They expose what they provide/require through interfaces, hiding their implementations to enhance reusability.

Each application component has *resource requirements* and *communication requirements* for its *connections* to other components. VM types have their *capacities*, *renting prices* and *communication capacities* to other VM types. This means that the resource capacities of a VM instance *must be larger than or equal to* the sum of resource requirements of the components it hosts. We call those resource capacities or requirements *dimensions*. Similarly, communication requirements between components must be inferior or equal to the communication capacities between the VM instances hosting them. We assume communication requirements and capacities can be expressed numerically.

Our hypothesis is that users describe communication constraints because they want a placement that respects their application *latency* requirements. Thus, to satisfy this constraint and to overcome the lack of available information about network topologies of public cloud providers, we introduce a flexible approach that allows an application designer to describe communication constraints using a less accurate view of the Cloud topology as well as a more accurate schema when in the context of private cloud providers.

Benefiting from this model, we propose an efficient and scalable heuristic that mixes graph clustering techniques and which is able to compute good quality placements very quickly for small to large scenarios. As this work considers *initial* placements, we do not assume *a priori* information concerning expected workload, renting times, or dynamic actors that would allow online modifications of the placement. This is left for future work. This paper extends our previous work [18], where we proposed bin packing based greedy heuristics to solve a *communication-oblivious* placement problem.

Section 2 deals with the state of the art. Section 3 presents our application and cloud models and details the proposed heuristic which is evaluated in Section 4. Finally, Section 5 concludes the paper and discusses future directions.

## 2 Related Work

We divide the related work into three groups based on the approach used to tackle the CAPDAMP and related *communication-aware* problems: exact approaches, meta-heuristic approaches, and heuristic based approaches. Then, we discuss them with respect to the CAPDAMP.

**Exact Algorithms:** In [19], a Mixed Integer Programming (MIP) is proposed for the placement of distributed applications on the Cloud. The objective is

to maximize availability by modeling fault-tolerance measures. Similarly, [10] proposes a MIP to minimize application downtime. Both approaches neither consider renting cost minimizations nor allow for more than two dimensions of interest. In [13], a very expressive MIP to compute the placement of services on multiple clouds is presented. Despite allowing for cost optimization, heterogeneous VM types, and resource constraints, it does not allow for an explicit description of communication constraints. Finally, in [12], a hierarchical approach to the process placement in multi-core clusters is presented. However, only the *communication problem* is considered and both processes and hosting machines are homogeneous, contratry to our work.

**Meta-heuristics:** In [3] and [20], the authors propose two very similar approaches based on genetic algorithms to calculate the placement of services on the Cloud targeting cost minimization while satisfying CPU, memory, disk and latency constraints. In [9], a simulated annealing based approach to the VM consolidation problem is presented. In the same topic, an ant colony algorithm for a multi-objective VM consolidation problem aiming at minimizing energy consumption and resource waste is described in [7]. In [6] another ant colony based approach for the VM consolidation problem is proposed.

**Heuristics:** A communication-aware greedy heuristic for calculating the task mapping on supercomputer clusters is presented in [4]. Using a max-clique based approach, [2] and [14] describe algorithms for the consolidation of VM types. An analogous problem is addressed in [16], which adds the challenge of having to place a virtual network aiming at satisfying resource and network constraints. Using a min cut approach, a hierarchical representation of the network and a graph modeling of the application, [15] tackles the traffic aware virtual machine placement on data centers. A hierarchical approach for the deployment of distributed scientific applications on the Cloud is presented in [5]. In [21], a graph matching algorithm based on a *graph query* approach for the service placement on the cloud is proposed. [8] presents a heuristic based on a relaxed MILP to compute a solution for a VM consolidation problem. In [11], an approach for placing services onto clouds while minimizing *communication* costs is proposed. Despite presenting a hierarchical cloud topology description and clustering heuristics similar to ours, the only considered resources is CPU. Communication constraints are viewed as soft constraints.

### 2.1 Discussion

As the CAPDAMP is NP-hard, using *exact algorithms* to calculate optimal placements is feasible only for very small problem instances. To overcome this limitation there is a plethora of more scalable approaches based mainly on meta-heuristics and heuristics. *Meta-heuristics* have their solution qualities proportional to the time given to process a problem. Hence, depending on problem size, using a meta-heuristic may still be unfeasible. Furthermore, as they are

generic tools, meta-heuristics tend to be very sensitive to parameter tuning specific for each scenario.

Other *heuristics* usually aim primarily at solving the graph partitioning (or communication constraint) problem letting the packing (or resource constraint problem) in second place. Graph-based modeling can efficiently describe communication constraints, however, describing at the same time resource constraints and renting costs tends to be more difficult. Thus, issues like VM heterogeneity, renting costs and multi-dimensionality are not addressed at the same time.

The main contribution of this paper is an efficient and scalable heuristic (cf. Section 3) which addresses the aforementioned problems. Using a graph clustering and multidimensional bin packing strategies, it manages to calculate good quality solutions very quickly, as described in Section 4.

## 3 The 2PCAP Heuristic

Before presenting the heuristic proposed in Section 3.1, we introduce the communication topology *models* for applications and multi-clouds upon which our heuristic strongly relies on.

**Multi-cloud Network Topology:** In this work, we model the *network* topology as a tree. It is a hierarchical approximation of intra cloud provider and long distance networks as well as an approximation of their inherent communication capacity *uncertainties*.

Figure 1(a) gives an example of this modeling. *Leaves* are sets of *rentable* resources, like VM types (or physical machines), that we call *machine groups*. An *inner node* $m$ of the multi-cloud tree models a level of *connection* between all machine groups available in the sub-tree having $m$ as root. The level of an inner node represents the *quality* of the machine group connection. In Figure 1(a), machine groups $m1$ and $m2$ are connected at levels 0, 1, and 2. Thus, their *connection qualities* may be 0, 1 or 2. Machine groups $m1$ and $m3$ have a connection quality 1. Resources in the same machine group are always connected with a connection quality equals to the level of the leaves. In Figure 1(a), all connections between VM types inside the same machine group have quality 3.

The concept of *connection quality* aims at characterizing the *latency* of a connection. The closest an internal node is to the leaves, the smallest the latency is. This is sufficiently general to describe detailed internal data center topologies as well as general Internet links. We suppose that this model is sufficient for describing VM localization (within the same data center, city, or country).

**Application Communication Topology:** In this work, we represent a component-based application as a *graph*. Components are nodes and connections between components are weighted edges. Weights are connection requirements, defined in terms of *connection quality* matching the multi-cloud topology. In Figure 2, components $c1$ and $c2$ communicate and require a connection quality of *at least* 2, while $c2$ and $c3$ require at least 1.

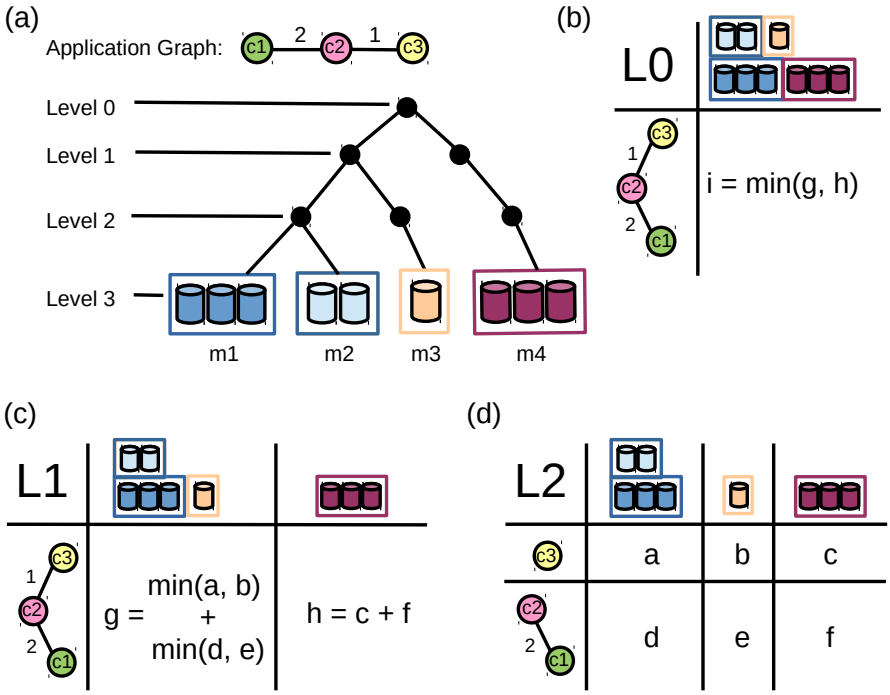


**Fig. 1.** Placement example.

### 3.1 Two Phase Communication Aware Placement Heuristic

We propose a divide-and-conquer heuristic called *Two Phase Communication Aware Placement Heuristic* (2PCAP) to calculate solutions for the CAPDAMP. 2PCAP, described in Algorithm 1, has two phases. i) It recursively *decomposes* components and machine groups into *subsets*, creating communication-aware *sub-placements*, until sub-placements can be calculated with *communication-oblivious* heuristics. ii) From the leaves to the root of the tree, sub-placements with best costs *compose* the solution for their parents.

**Phase 1 – Decomposition:** A *component subset* $i_\ell$ is a set of nodes from a connected subgraph from the application graph. Component subsets have the property that every connection between its components has a communication quality requirement superior or equal to $\ell$, where $0 \leq \ell < H$ and $H$ is the height of the multi-cloud tree. A *machine group subset* $s_\ell$ contains machine groups from sub-trees of the multi-cloud tree topology. All machine groups contained in the same subset are connected with connection quality superior or equal to $\ell$.

The process that *generates* component and machine group subsets is called *decomposition*. Given a level $\ell$, machine group subsets are generated through the gathering of leaves whose subtree roots are in level $\ell$. Component subsets

**Algorithm 1** Pseudo-code of 2PCAP.

**Input:** $level, comp\_subset, mg\_subset$
**Output:** $min\_cost\_plac$

```
1: min_cost_plac ← ∞
2: if is_calculated(comp_subset ⇒ mg_subset) then
3: |   return plac(comp_subset ⇒ mg_subset)
4: else if level = l_max then
5: |   calculate(comp_subset ⇒ mg_subset)
6: |   return plac(comp_subset ⇒ mg_subset)
7: else if level < l_max then
8: |   if size(decompose(mg_subset, level)) = 1 then
9: |   |   plac ← null
10: |   |   for cs in decompose(comp_subset, level) do
11: |   |   |   temp_plac ← 2PCAP(level + 1, cs, mg_subset)
12: |   |   |   plac ← compose(plac + temp_plac)
13: |   |   min_cost_plac ← plac
14: |   else if size(decompose(mg_subset, level)) > 1 then
15: |   |   plac ← null
16: |   |   for cs in decompose(comp_subset, level) do
17: |   |   |   min_plac ← null
18: |   |   |   for ms in decompose(mg_subset, level) do
19: |   |   |   |   temp_plac ← 2PCAP(level + 1, cs, ms)
20: |   |   |   |   if cost(temp_plac) < min_plac then
21: |   |   |   |   |   min_plac ← temp_plac
22: |   |   |   plac ← compose(plac + min_plac)
23: |   |   min_cost_plac ← plac
24: |   for ms in decompose(mg_subset, l_max) do
25: |   |   temp_plac ← 2PCAP(l_max, comp_subset, ms)
26: |   |   if cost(temp_plac) < cost(min_cost_plac) then
27: |   |   |   min_cost_plac ← plac
28: return min_cost_plac
```

are connected sub-graphs resulting from the removal of all connections requiring connection qualities inferior to $\ell$ from the original application graph. $\mathcal{I}_\ell$ and $\mathcal{S}_\ell$ are the sets containing, respectively, all components and machine groups subsets *constructed on* level $\ell$.

In Figures 1(b), 1(c) and 1(d) there are examples of machine group and component decompositions. Table names (L0, L1 and L2) refer to the level of decomposition, component subsets ($\mathcal{I}_\ell$) are represented in the left and machine group subsets ($\mathcal{S}_\ell$), in the upper part.

A *sub-placement* is the placement of a subset of components on a subset of machine groups. It also aims at minimizing VM renting costs while satisfying resource and communication constraints. Given a level $\ell$, $i_\ell \in \mathcal{I}_\ell$ and $s_\ell \in \mathcal{S}_\ell$, the sub-placement of $i_\ell$ on $s_\ell$ can only be computed if it is a *bottom sub-placement* or if the sub-placements generated by the decomposition of $i_\ell$ and $s_\ell$ were *computed*.

A *bottom sub-placement* is a sub-placement that can be computed by communication oblivious heuristics while satisfying communication quality require-

ments. Hence, any pair of VM types from machine groups contained in $s_\ell$ will satisfy the communication requirements from any pair of components from $i_\ell$.

Let $l^{max}$ be the highest connection quality requirement present in the component graph. Observe that every sub-placement of $i_\ell \in \mathcal{I}_\ell$ on $s_{\ell^{max}} \in \mathcal{S}_{\ell^{max}}$ is a valid bottom sub-placement. Hence, there is no reason to continue the decomposition process beyond $l^{max}$.

**Phase 2 – Composition:** Bottom sub-placements are calculated by efficient communication-oblivious heuristics for the multi-dimensional bin packing problem presented in a previous work [18]. Once all necessary bottom sub-placements are calculated, 2PCAP starts the process of *composition of sub-placements*. The objective is to choose, at each composition step, the less expensive sub-placements. Given the set $\mathcal{I}'_{\ell+1}$ containing all component groups decomposed from $i_\ell \in \mathcal{I}_\ell$ and the set $\mathcal{S}'_{\ell+1}$ decomposed from the machine group $s_\ell \in \mathcal{S}_\ell$, let $u^{is}_{\ell+1}$ be the sub-placement of $i_{\ell+1}$ on $s_{\ell+1}$. Thus, the solution for the sub-placement of $i_\ell$ on $s_\ell$ is one of the following:

Case 1: $u^{is}_{\ell+1}$, if $\mathcal{S}_{\ell+1} = \mathcal{S}_\ell$ and $\mathcal{I}_{\ell+1} = \mathcal{I}_\ell$.
Case 2: $\sum_{i \in \mathcal{I}_{\ell+1}} u^{is}_{\ell+1}$ for $s \in \mathcal{S}_{\ell+1}$, if $|\mathcal{S}_{\ell+1}| = |\mathcal{S}_\ell|$ and $|\mathcal{I}_{\ell+1}| > |\mathcal{I}_\ell|$;
Case 3: $\sum_{i \in \mathcal{I}_{\ell+1}} min(u^{is}_{\ell+1},\ \forall s \in \mathcal{S}_{\ell+1})$, if $|\mathcal{S}_{\ell+1}| > |\mathcal{S}_\ell|$;

In Case 1, The decomposed subset of components and machine groups are identical to the original subsets. Hence, $u^{is}_\ell = u^{is}_{\ell+1}$. This is described in lines 2 and 3 from Algorithm 1. In Case 2, the decomposed subset of machine groups is identical to the original, but this is not true for the decomposed component subset. In this case, 2PCAP composes the $|\mathcal{I}_{\ell+1}|$ sub-placements on $s_{\ell+1}$. Sub-placements $c$ and $f$ (cf. Figure 1(d)) compose sub-placement $h$ (cf. Figure 1(c)). This situation is described between lines 8 and 13 from Algorithm 1. In Case 3, when $|\mathcal{S}_{\ell+1}| > |\mathcal{S}_\ell|$ machine groups are decomposed in more than one subset. Thus, for each decomposed component subset there are $|\mathcal{S}_{\ell+1}|$ possible sub-placements, from which, only the less expensive one is used in the composition process. Sub-placement $i$ (cf. Figure 1(b)) is composed by sub-placements $g$ and $h$ (cf. Figure 1(c)). Furthermore, sub-placement $g$ (cf. Figure 1(c)) is composed by sub-placements $a$, $b$, $d$ and $e$ (cf. Figure 1(d)). This can be observed between lines 14 and 23 from Algorithm 1.

### 3.2 Discussion

The 2PCAP heuristic does not compute, during the decomposition phase, all possible sub-placements. Doing this would result in a factorial complexity which would lead to prohibitive execution times for large problems. To further explore the solution space without increasing too much the time complexity, 2PCAP computes the sub-placement of every generated subset which is not part of a bottom sub-placement on machine group subsets generated at level $l^{max}$ (cf. Lines 24 to 27 of Algorithm 1).

The complexity of 2PCAP is dominated by decomposition operations (*decompose* function) and the computation of placements (*plac* function). Let $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{T}$ be the sets of components, sites and VM types, respectively. Decomposition operations have a $O(|\mathcal{I}|^3 + |\mathcal{I}| \times |\mathcal{S}|^2 \times log|\mathcal{S}|)$ complexity while the computation of placements has $O(|\mathcal{S}| \times |\mathcal{T}|log|\mathcal{T} \times |\mathcal{I}|^2)$.

## 4 Evaluation

As the CAPDAMP is NP-Hard, we divide the evaluation process in two steps. First, using *small problem instances* and a MIP solver, we compare 2PCAP solutions to optimal ones. Then, we compare 2PCAP on *medium* and *large* problem instances using meta-heuristics and a relaxed version of the CAPDAMP as baseline algorithms.

### 4.1 Methodology

An *experiment* is the resolution of a set of placement *problem instances* by a set of algorithms within a given *time*. Each problem instance has seven parameters: the number $n_d$ of considered resources or *dimensions*, the number $n_c$ of components, the number $n_v$ of VM types, the number $n_s$ of sites, the height $h_t$ of the multi-cloud tree, the topology $t_c$ of the component-based application and the multi-cloud tree connection schema $x_t$.

Experiments are organized in three *experiment classes*, namely *A*, *B*, and *C*. Small, and thus easier to solve, problem instances compose Class A; medium-sized problem instances are present in Class B, and, finally, large problems form Class C. Table 1 details the range of problem instance parameters that define each class and the total of generated problem instances per class.

| class | $n_d$ | $n_c$ | $n_v$ | $n_s$ | $h_t$ | $t_c$ | $x_s$ | # exps |
|---|---|---|---|---|---|---|---|---|
| A | 4 | $3, 5, 7, 10$ | $100, 250, 500, 700$ | $25, 50, 100$ | $3, 5$ | $l, s, f, r$ | $u$ | 384 |
| B | 5 | $10, 20, 30, 40, 50$ | $500, 1k, 1.5k, 2k$ | $100, 300, 500$ | 5 | $l, s, f, r$ | $d, a, u$ | 720 |
| C | 6 | $60, 80, 100, 120, 140$ | $2.5k, 5k, 7.5k, 10k$ | $500, 750, 1k$ | 7 | $l, s, f, r$ | $d, a, u$ | 720 |

**Table 1.** Parameters of experiment classes. Column $t_c$ indicates the application topologies: line ($l$), star ($s$), full connected ($f$), or random ($r$). Column $x_s$ indicates the multi-cloud tree connection schemas: distant($d$), agglomerate ($a$), or uniform ($u$).

Component requirements and VM capacities are pseudo-random values, picked uniformly from pre-defined intervals (Table 2). We consider that VM types are distributed equally among the sites. We generate three different component communication patterns: *distant*, *agglomerated*, and *uniform*. The difference between them is the probability of connecting two or more subtrees. The *distant* pattern has higher probability to connect subtrees near the root; *agglomerated* gives

| Dimension | (i) | (ii) | (iii) | (iv) | (v) | (vi) |
|---|---|---|---|---|---|---|
| **Requirements** | $800 \sim 3k$ | $1 \sim 16$ | $1 \sim 32$ | $50 \sim 3.5k$ | $5 \sim 30$ | $1 \sim 8$ |
| **Capacities** | $1k \sim 3.5k$ | $2 \sim 32$ | $2 \sim 40$ | $150 \sim 4k$ | $10 \sim 80$ | $1 \sim 16$ |

**Table 2.** Intervals of dimension data generation.

higher connection probabilities to subtrees near the leaves and the uniform schema gives the same connection probability $\left(\frac{1}{h_t}\right)$ to every subtree.

The four component-based application topologies we consider are *line*, *star*, *full connected* (cf. Figure 2), and *random*. In the random schema, a pair of components is connected with a probability of 50%. Communication requirements from component connections are pseudo-random integers picked uniformly between 0 and $h_t - 1$.

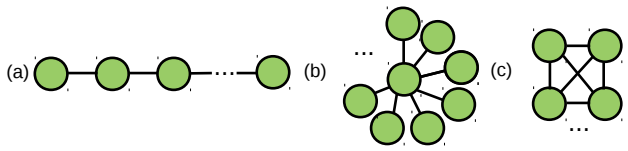


**Fig. 2.** Schemas of part of the generated application topologies. (a) *line*, (b) *star* and (c) *full connected*.

Renting prices depend on the resource dimensions of each VM. Let $c^*_{t,d}$ be the ratio $\frac{c_{t,d}}{max_d}$ between the capacity $c_{t,d}$ of dimension $d$ from VM type $t$ and $max_d$ the maximum value for dimension $d$ (cf. Table 2). Each dimension is multiplied by a coefficient to create scenarios where some dimensions are more expensive than others. Hence, the price of a VM type $p_t$ is $\alpha + \beta + \gamma + \delta + \epsilon + \zeta$, where $\alpha = c^*_{t,1} \times random(1,3)$, $\beta = c^*_{t,2} \times random(8,20)$, $\gamma = c^*_{t,3} \times random(5,8)$, $\delta = c^*_{t,4} \times random(10,15)$, if $c^*_{t,4} \leq 500$, otherwise $\delta = c^*_{t,4} \times random(20,25)$, $\epsilon = c^*_{t,5} \times random(5,10)$, and $\zeta = c^*_{t,6} \times random(2,5)$.

The 2PCAP algorithm is implemented in Python. Experiments were conducted on Dell PowerEdge R630 2.4GHz (2 CPUs, 8 cores) nodes from the *Parasilo* and *Paravance* clusters of the *Grid'5000* experimental platform[1].

### 4.2 2PCAP Performance on Small Problems

In this section we use the *SCIP* solver [1], a framework for constraint integer programming and branch-cut-and-price, together with an optimization formulation of the CAPDAMP to generate a set of optimal solutions which will be compared to solutions computed by 2PCAP. Problems from experiment Class A (Table 1) were used and SCIP was given 24 hours to solve each one of them.

[1] cf. `https://www.grid5000.fr`

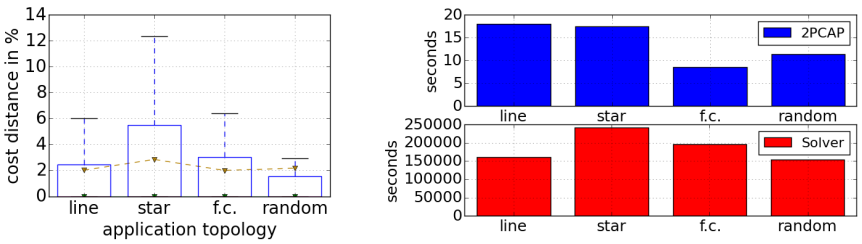


**Fig. 3.** (Left) Cost distances as percentage of 2PCAP solutions compared to optimal solutions aggregated by application topology type. The green solid line is the median and the brown dashed line is the average. (Right) Sum of execution times in seconds from 2PCAP and SCIP solver aggregated by application topology – SCIP: average 4180s, median 1800s. 2PCAP: average 0.1s, median: 0.08s.

SCIP solver was able to solve only around 48% of Class A problem instances in time, *i.e.,* 180 problem instances. Figure 3 illustrates the *cost distance* from solutions computed by 2PCAP to the optimal ones as a percentage of the latter for problem instances successfully solved by SCIP. Cost distances are grouped by application topology.

In Figure 3(Left) we can see that cost distances vary between 0% and at most 12.3%. The median is always 0% and the average between 2% and 3%. Figure 3(Right) complements this data. It depicts the sum of the execution times in seconds that each approach used to calculate the 48% of Class A problem instances solved by SCIP, grouped by application topology schema. While 2PCAP takes some seconds to solve all problems, the solver's execution time is in the scale of days. Hence, in spite of being much faster than the solver, 2PCAP manages to produce a solution at most around 12% worse than the optimal and, in the median, the solutions are optimal.

### 4.3 2PCAP Performance on Large Problems

The evaluation of 2PCAP on large problems cannot rely on using a solver to generate optimal solutions. It is necessary to use scalable baseline algorithms.

Heuristics presented in Section 2 cannot handle the complexity of CAPDAMP. Hence, as a first approach, we implement a Simulated Annealing (SA) meta-heuristic for the CAPDAMP as a baseline algorithm. We used Python module *Simanneal* [17] and problem instances from Classes B and C. The meta-heuristic is initialized with a random – not necessarily valid – placement. Despite the timeout of 1 hour per problem instance, SA managed to calculate a solution for only around 10% of problem instances. As CAPDAMP's search space is very large, SA would need more time to be able to produce more solutions.

Running SA with an initial solution computed by 2PCAP shows how much SA can improve a 2PCAP solution in one hour. Figure 4 (Left) illustrates this metric for Class C problems. SA managed to improve the solutions by at most 9%

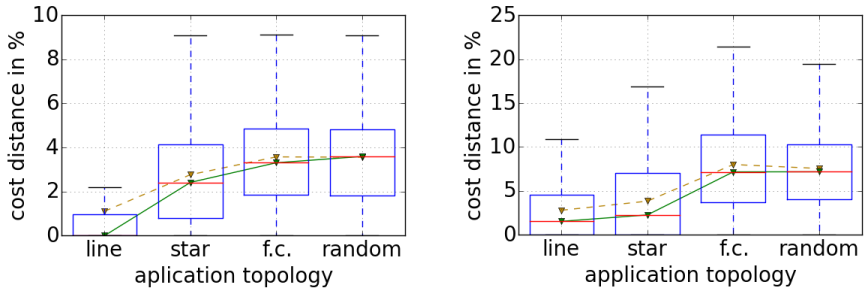


**Fig. 4.** (Left) Improvement of 2PCAP solutions by SA for Class C. (Right) Cost distances between 2PCAP and lower bound for Class C. The green solid line is the median and the brown dashed line is the average.

and the median is always bellow 4%. This small improvement is a good indicator of 2PCAP's solution qualities. Due to space limitations, we do not illustrate the same metric for Class B problems, however the observed curves are very similar: the largest improvement is around 9% and the median is always bellow 2%.

In a second approach, we compute baseline solutions obtained through the relaxation of CAPDAMP's communication constraints and compare them to 2PCAP solutions. CAPDAMP is, thus, reduced to a cost-aware multi-dimensional bin packing placement problem which is calculated by an adapted SA metaheuristic initialized with the less expensive placement among those calculated by 2PCAP and other efficient heuristics [18]. Figure 4 (Right) illustrates the evolution of cost distances between 2PCAP and SA solutions for Class C problems. Cost distances vary between 0% and around 22% and the median is always bellow 10%. Due to space constraints, we do not plot this metric for Class B problems, nevertheless, we observe a similar pattern: cost distances vary between 0% and 30% and the median is always bellow 10%. The consistent distance to baseline solution costs is a good indicator of the 2PCAP's solution quality. Concerning 2PCAP execution times for Classes B and C, the average was around 11 seconds and the median around 3 seconds for each problem instance. At most, 2PCAP took around 110 seconds to calculate a placement.

The results presented in this section indicate that, even on large scenarios, 2PCAP manages to quickly calculate compatible or better solutions than those calculated by SA.

## 5 Conclusion and Future Work

In this paper we presented an approach to calculate initial placements for component-based applications with the objective of minimizing costs while satisfying resource and communication constraints. This approach is based on a hierarchical model of the cloud topology which allows the introduction of latency requirements despite the uncertainties inherent to cloud networks, mainly due to virtu-

alization. This model is used by 2PCAP, an efficient heuristic whose evaluation shows its capability of producing good quality solutions very quickly.

Future work aims to go beyond the initial placement by adding the notion of application reconfiguration and, consequently, modeling the migration of virtual machines. We also plan to extend the placement heuristics to support applications described with more abstract component models, including, for example, concepts such as cardinality, hierarchy, genericity, etc.

## Acknowledgment

All experiments were carried out using the Grid'5000 testbed, supported by a group hosted by Inria and including CNRS, RENATER, and several Universities as well as other organizations (cf. `https://www.grid5000.fr`). This work was partially supported by the PaaSage (FP7-317715) EU project.

## References

1. Achterberg, T.: SCIP: Solving Constraint Integer Programs. Mathematical Programming Computation (2009)
2. Biran, O., Corradi, A., Fanelli, M., Foschini, L., Nus, A., Raz, D., Silvera, E.: A Stable Network-Aware VM Placement for Cloud Systems. In: CCGrid (2012)
3. Chen, W., Qiao, X., Wei, J., Huang, T.: A Profit-Aware Virtual Machine Deployment Optimization Framework for Cloud Platform Providers. In: CLOUD (2012)
4. Deveci, M., Kaya, K., Uçar, B., Catalyurek, U.V.: Fast and High Quality Topology-Aware Task Mapping. In: IPDPS (2015)
5. Fan, P., Chen, Z., Wang, J., Zheng, Z., Lyu, M.R.: Topology-Aware Deployment of Scientific Applications in Cloud Computing. In: CLOUD (2012)
6. Ferdaus, M.H., Murshed, M., Calheiros, R.N., Buyya, R.: Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. In: Europar (2014)
7. Gao, Y., Guan, H., Qi, Z., Hou, Y., Liu, L.: A Multi-objective Ant Colony System Algorithm for Virtual Machine Placement in Cloud Computing. Journal of Computer and System Sciences (2013)
8. Gu, L., Zeng, D., Guo, S., Xiang, Y., Hu, J.: A General Communication Cost Optimization Framework for Big Data Stream Processing in Geo-Distributed Data Centers. IEEE Transactions on Computers (2016)
9. Hyser, C., Mckee, B., Gardner, R., Watson, B.J.: Autonomic virtual machine placement in the data center. Tech. Rep. HPL-2007-189, HP Laboratories (2007)
10. Jammal, M., Kanso, A., Shami, A.: High Availability-Aware Optimization Digest for Applications Deployment in Cloud. In: ICC (2015)
11. Jayasinghe, D., Pu, C., Eilam, T., Steinder, M., Whally, I., Snible, E.: Improving Performance and Availability of Services Hosted on IaaS Clouds with Structural Constraint-Aware Virtual Machine Placement. In: SCC (2011)
12. Jeannot, E., Mercier, G., Tessier, F.: Process Placement in Multicore Clusters: Algorithmic Issues and Practical Techniques. IEEE Transactions Parallel Distributed Systems (2014)
13. Lucas-Simarro, J.L., Moreno-Vozmediano, R., Montero, R.S., Llorente, I.M.: Scheduling Strategies for Optimal Service Deployment across Multiple Clouds. Future Generation Computer Systems (2013)

14. M. Alicherry, T.L.: Network Aware Resource Allocation in Distributed Clouds. INFOCOM (2012)
15. Meng, X., Pappas, V., Zhang, L.: Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement. In: INFOCOM (2010)
16. Nonde, L., El-Gorashi, T.E.H., Elmirghani, J.M.H.: Energy Efficient Virtual Network Embedding for Cloud Networks. Journal of Lightwave Technology (2015)
17. Perry, M.: Simanneal: Python Module for Simulated Annealing Optimization, `https://github.com/perrygeo/simanneal`
18. Silva, P., Perez, C., Desprez, F.: Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds. In: CCGrid (2016)
19. Spinnewyn, B., Braem, B., Latre, S.: Fault-Tolerant Application Placement in Heterogeneous Cloud Environments. In: CNSM (2015)
20. Yusoh, Z.I.M., Tang, M.: Clustering Composite SaaS Components in Cloud Computing using a Grouping Genetic Algorithm. In: CEC (2012)
21. Zong, B., Raghavendra, R., Srivatsa, M., Yan, X., Singh, A.K., Lee, K.W.: Cloud Service Placement via Subgraph Matching. In: ICDE (2014)