Virtual Machine Placement for Hybrid Cloud using Constraint Programming

Hélène Coullon, Guillaume Le Louët, Jean-Marc Menaud

Virtual Machine Placement for Hybrid Cloud using Constraint Programming

Hélène Coullon, Guillaume Le Louet, Jean-Marc Menaud
Ascola, IMT Atlantique DAPI, Inria, LS2N
Nantes, France
Email: helene.coullon@imt-atlantique.fr, guillaume.lelouet@gmail.com, jean-marc.menaud@imt-atlantique.fr

***Abstract*—Cloud computing is the widely spread paradigm of utility-computing that offers an "on-demand" internet-based access to configurable resources available within data centers. On one hand, public Cloud providers are well suited for highly available access to IT resources (infrastructure, platform and software), for sporadic use, or for elastic demands. On the other hand, private clouds could sometimes be preferred for security or privacy reasons, or for cost reasons due to a high frequency usage of services. However, in many cases a choice between public or private clouds does not fulfill all requirements of companies, and hybrid cloud infrastructures should be preferred. A hybrid cloud solution could, for example, answer sudden workload increase in private clouds, security or fault tolerance requirements, or even latency issues thanks to data-locality. Solutions have already been proposed to address hybrid cloud infrastructures, however most of the time the placement of a distributed software on such infrastructure has to be indicated manually. For this reason, the automation of software deployment on hybrid clouds is still under research. In this paper we propose new specific placement constraints and objectives adapted to hybrid clouds infrastructures within our placement solution, namely OptiPlace, and we address this problem through constraint programming. Furthermore, we evaluate the expressivity and performance of the proposed solution on a real case study.**

***Keywords*-Hybrid Cloud; Virtual Machines Placement; Constraint Programming;**

I. Introduction

Cloud computing is the widely spread paradigm of utility-computing that proposes an "on-demand" internet-based access to configurable resources available within data centers. One of the great advantage of Cloud computing is being able to use as many resources as needed at a given time, namely elasticity, while asking minimal administration efforts compared to internal private servers. The economic model under the Cloud paradigm is often called *pay-as-you-go*.

Most of the time, when talking about Cloud computing, public Cloud providers are considered. Public providers enable simple access to resources for companies and users who have sporadic or elastic demands and who need high availability of services. However, sometimes for security or privacy reasons, or sometimes because of the high usage frequency of the Cloud (and thus for cost reasons), companies prefer to handle their own private Cloud solution. As a result, employees can use internal servers the same way as a public Cloud. Of course, this solution re-introduces administration costs, that could however be offset by computation and storage cost savings. Moreover, the elasticity offered by public Clouds is more limited within private Clouds (as resources are more limited). Thus, the choice between private or public Cloud is specific to each company, a trade-off has to be found between cost, security, performance etc.

In many cases, though, a choice between public or private Clouds does not fulfill all requirements of companies and an hybrid solution should be preferred. First, a hybrid Cloud solution can answer sudden workload increase that exceeds capacities of the private Cloud. In such case, an occasional demand for external private or public Clouds is an interesting solution. Second, if some software components and specific data may require to be hosted locally onto the private Cloud of the company, others may on the contrary need to be hosted into a distant server, for example to stay close to end-users for latency reasons, or because of external data requirements, or finally to prevent software and systems from widespread failures etc.

Solutions have been proposed to be able to federate more than one Cloud provider (public or private) by collaborations of IaaS systems [1], [2]. Thus, the deployment of a distributed software on multiple Cloud providers is partly solved by manually indicating where to place each software component or service. However, the complete automation of this deployment process, is still under research. One of the most important sub-process of deployment automation is to handle the automation of placement problems hided behind IaaSes. A placement problem appears when you try to place n virtual machines (or container), by answering their associated placement constraints, onto m physical machines with their own capacities. Moreover, a placement problem often answers one or multiple objective functions. This is a NP-hard multi-dimensional and multi-objective bin-packing problem. Solutions has been proposed to solve such placement problem [3], [4] thus automating services and distributed software deployment. However, as far as we know, specific placement constraints for hybrid Cloud have not been studied yet.

In this paper, we use the flexible placement framework OptiPlace [5] to offer a new set of constraints and objective functions for the specific case of hybrid Cloud and Cloud federation. OptiPlace uses constraint programming in a flexible way by proposing software engineering capabilities such

as modularity, separation of concerns and maintainability. Contributions of the paper are:

- A generic infrastructure model to tackle hybrid Clouds and Cloud federation;
- A new set of constraints into OptiPlace for the specific context of hybrid Clouds and Cloud federation;
- An evaluation of the new placement constraints on the real case study TrustyDrive, a distributed storage system.

The rest of this paper is organized as follows: Section II introduces context and related works, Section III gives an overview of OptiPlace and its view mechanism, Section IV presents the contribution of the paper and Section V its evaluations. Finally, Section VI concludes the work and opens to future challenges.

II. Context and Related Work

Many applications can benefit from hybrid Clouds. We can classify them into four classes that can overlap each other. The first class is the set of applications that seasonally or punctually need to increase its capacity. As the need is brief companies will not buy additional resources for their own private Cloud but should instead use additional resources from external public Cloud providers. The second class is the set of applications that need to duplicate their services, not because of higher workloads, but for reliability, *i.e.*, to prevent from failures. For example, *ever-running* applications such as commercial websites or sandboxes must always be available.A third class can be considered for geo-distributed applications such as *smart-** applications. In this case, by selecting one Cloud provider or another (following proximity constraints), hybrid Clouds can help to reduce latencies. Finally, the fourth class to consider is the set of applications that wants to place data or computations within a specific country, region or data center for security and data privacy reasons.

Because of such applications, hybrid Clouds have become an active research domain. Currently, many issues need to be solved to be able to reach the ideal hybrid Cloud infrastructure. For example, in [6] authors tackle the problem of heterogeneous virtualization techniques in hybrid Cloud infrastructures. In [7] is proposed a solution to be able to get a failure-aware resource provisioning in the context of a hybrid Cloud infrastructure. Another research problem is to solve scheduling problems to minimize the cost for companies by using hybrid Clouds [8].

Recent IaaS managers allow users to move their local virtual machines (VMs) on external Clouds. First, VMWare proposes *vCloud Air*, a public Cloud that extends a company's *vSphere* private infrastructure [1]. In this solution, VMWare VMs can be migrated from or to the public Cloud. HP proposes the *Helion* [2] IaaS manager. Helion is based on OpenStack and contains the *Eucalyptus* module to integrate VMs from Amazon's AWS public Cloud. Finally, Fujitsu offers *Hybrid Cloud Services* [3] that can migrate VMs from their private Cloud solution to the Microsoft Windows Azure public Cloud. These hybrid Cloud managers provide integrated solutions to migrate VMs from private to public Cloud or the opposite. However, none provides an automated deployment and migration process. When deploying a complex distributed application onto a large Cloud infrastructure, though, a fully automated process is needed by users and administrators. One important challenge of this automation is to optimize services placement, such that resources usage, and costs are themselves optimized.

As already mentioned a placement problem with multiple variables and objectives is NP-hard which leads to two different techniques to solve it. If the problem is small enough it can be solved within an acceptable time by exact solutions such as linear programming [9] or constraint programming [3]. If the problem is larger or have to be solved faster, however, heuristics and meta-heuristics should be preferred [4], [10].

In this paper we use the OptiPlace [5] placement framework based on the constraint programming solver Choco [4]. The Choco solver has previously been used to solve placement problems, as for example in BtrPlace [3]. However, and as it will be explained in Section III, OptiPlace has been designed to enhance software engineering capabilities to easily add new problems to the initial basic one, statically or on the fly. This makes OptiPlace different from existing placement frameworks. For example, OptiPlace has been previously used to easily add placement problems related to energy-saving constraints [5]. In this paper are proposed specific constraints for hybrid Clouds. These new elements have been added within OptiPlace. As far as we know, no existing solution solve such placement problems by using constraint programming.

III. OptiPlace

OptiPlace is part of a broader chain of frameworks developed during the Hosanna project. The complete chain of frameworks, namely Entropy, is able to deploy and reconfigure on-the-fly a distributed application onto an hybrid infrastructure. In this paper, we focus on the presentation and the evolution of the OptiPlace framework responsible for placement problems within Entropy.

OptiPlace [5] is an independent flexible framework to solve placement problems for data centers. Thus, it could be used by any Infrastructure-as-a-Service system to optimize the placement of virtual machines (or containers) onto physical machines. For example, it could be used inside OpenStack or by cloud providers such as Amazon Web

[1] http://www.vmware.com/fr/products/vCloud-suite.html

[2] http://www8.hp.com/fr/fr/Cloud/helion-eucalyptus-overview.html

[3] https://www.fujitsu.com/global/services/application-services/application-development-Integration/hybridCloud/

[4] http://www.choco-solver.org/

Services (AWS EC2), VMware, Google Compute Engine etc. Even so, OptiPlace is more suitable to handle private clouds.

OptiPlace is based on the constraint solver Choco [11], written in java. As with any other constraint solver, a Choco problem is composed of *variables* that represent unknowns of the problem, and *constraints* applied onto variables. A solution affects values to variables while verifying constraints. With variables and constraints only, solvers will stop as soon as a valid solution is found (if it exists). Sometimes though, better solutions could be found according to some *objectives*. This is the third element of a constraint program. Most of the time objectives try to maximize or minimize a function that is computed from the values given to variables. Finally, in Choco it is also possible to provide exploration strategies, or exploration heuristics, for a given problem thanks to a built-in API. Such strategies are very important to speedup the problem solving.

One of the strengths of constraint programming is that any initial constraint program can be enriched afterwards by additional variables, constraints and objectives. For this reason, constraint programming facilitates flexibility. OptiPlace makes use of this flexibility in a more specialized way, for Cloud Computing and Virtual Machines placement problems. Actually, OptiPlace proposes a core constraint program and a *view* mechanism to add or remove sub-problems to the core one, on the fly. These mechanisms of OptiPlace are detailed in the rest of this section.

A. Core problem of OptiPlace

The core problem of OptiPlace is a reconfiguration problem inherited from Entropy [12]. This core problem is to allocate a set of virtual machines to a set of hosts, taking into account (or not) the current placement (the current execution state) of a subset of these virtual machines. This problem is modeled as follows: the set of all virtual machines to allocate is denoted $\mathcal{V}$; a data center is composed of a set of nodes, *i.e.*, physical machines, denoted $\mathcal{N}$; each node $n_i \in \mathcal{N}$ is associated to the vector $H_i =< h_{i1}, \ldots, h_{ij}, \ldots, h_{i|\mathcal{V}|} >$ where $h_{ij} = 1$ if the virtual machine v_j is hosted by n_i, $h_{ij} = 0$ otherwise.

The set of types of resource capacities (*i.e.*, memory, cpu and disk for example) is denoted $\mathcal{C}$. Its size is $|\mathcal{C}|$. For each resource capacity type $k \in \mathcal{C}$ (for example cpu), two vectors are associated:

- C_k denotes the vector representing the resource capacity of type k associated to each node $n_i \in \mathcal{N}$, the size of C_k is $|C_k| = |\mathcal{N}|$, and we denote as $C_k(i)$ the available resource capacity of type k for the node $n_i \in \mathcal{N}$;
- R_k denotes the vector representing the needed resource capacity of type k associated to each virtual machine $v_i \in \mathcal{V}$, the size of R_k is $|R_k| = |\mathcal{V}|$.

In this core problem, there are as many constraints as the number of types of resource capacities $|\mathcal{C}|$. Each of these constraints is defined as follows, for a capacity k:

$$R_k.H_i \leq C_k(i) \forall n_i \in \mathcal{N}. \quad (1)$$

This means that for a given type of capacity k, the overall capacity needed by the virtual machines hosted onto the node n_i must not exceed the maximum capacity of n_i itself.

Unlike Entropy, OptiPlace takes into account as many resource capacities as needed by the cloud administrator. Moreover, unlike Entropy, OptiPlace does not have a default objective. This means that the Choco solver will stop as soon as a solution is found in the core Constraint Satisfaction Problem (CSP).

B. Views mechanism

From the cloud administrator point of view, activating a *view* is a way to add new variables, constraints and objectives to the core problem of OptiPlace. Basic views are already proposed inside OptiPlace, however it is also possible to define new ones by using the OptiPlace-API. This API is important to facilitate the addition of sub-problems to the core one without knowing the Choco solver. Thus, the view mechanism offers an abstraction over the solver.

When a placement problem is solved with OptiPlace, more than one view can be activated. Moreover, a mechanism of dependencies between different views is also possible.

One of the views proposed inside OptiPlace is the *High Availability* view (HA). HA contains basic placement constraints, most of them inherited from Entropy. Among these constraints are the one called *spread* that will be used in this paper. The spread constraint offers a way to specify that a subset of virtual machines $\mathcal{V}_{sp} \subset \mathcal{V}$ must be spread among available servers, *i.e.*, must be placed on different servers. This constraint can be formalized as follows:

$$\left| \bigcup_{\substack{n_i \in \mathcal{N} \\ h_{ij}=1 \\ v_j \in \mathcal{V}_{sp}}} n_i \right| = |\mathcal{V}_{sp}| \quad (2)$$

In other words,

$$\forall (v_j, v_l) \in \mathcal{V}_{sp}^2 \mid h_{ij} = 1 \wedge h_{kl} = 1 \Rightarrow n_i \neq n_k \quad (3)$$

Many others constraints exist in HA, for example *lonely* guarantees that a given virtual machine is the only one hosted on a server.

Another view of OptiPlace used in this paper is the *Host Cost* view (HC). HC proposes to add specific variables representing the cost of using a given node of $\mathcal{N}$ as a host for one or several virtual machines. The HC view also adds an objective function to minimize the overall cost of the

placement solution. If χ denotes the vector of cost for all nodes $n_i \in \mathcal{N}$, the cost function to minimize is denoted

$$X = \sum_{i \in \mathcal{N}} u_i \text{ where } u_i = \begin{cases} \chi(n_i) & \exists v_j \in \mathcal{V} \mid h_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Other views than HA and HC have also been added to OptiPlace. For example, in [5] a specific view to optimize data centers energy consumption has been proposed.

IV. Cloud federation with OptiPlace

Until now, OptiPlace has been built to handle servers managed by a single owner (or Cloud provider). Thus, it was not possible to take into account external resources from other Cloud providers, excluding Cloud federation as well as hybrid Cloud solutions.

In this paper, three new infrastructure concepts are introduced, namely *extern*, *site* and *tag*, to be able to manage external nodes as well as different sites and tags, *i.e.*, groups of servers. Moreover, a new view is added to OptiPlace, namely the *SkyPlace* view (SP), to add specific constraints linked to hybrid Cloud, and more widely to Cloud federation: OnSite, OffSite, Near and Far.

A. Infrastructure modeling

Extern node: the set of external nodes is denoted $\mathcal{E}$. Such as an internal node $n_i \in \mathcal{N}$, an external node is associated to a set of resource capacities $\mathcal{C}$. However, unlike internal nodes, an *extern* is a type of node provided by an external Cloud provider (public or private). It is not limited to one ocurrence as well as not associated to one given hardware resource. Thus, one can ask as many external nodes of a given type as required, without limitation. As a result, the constraint of Equation (1) is not associated to external nodes. This concept is similar to a *flavor* in OpenStack or a type of virtual machine to rent in AWS. Without any specific constraints or objective functions, that could preferably choose internal nodes for cost reasons, or on the contrary external nodes because of energy limitations, an external node is considered exactly as any other node but is unlimited as far as it is needed. This additional type makes possible specific constraints or objectives for Cloud federation and hybrid Clouds. One can note that the $Spread$ constraint applied to external nodes means that two different types of external nodes e_1 and $e_2 \in \mathcal{E}$ should be used.

Site: a site is a subset of servers $S \subset (\mathcal{N} \cup \mathcal{E})$. Each node $n \in (\mathcal{N} \cup \mathcal{E})$ is associated to a single site. Thus for two nodes n_1 and n_2, if $site(n_1) \neq site(n_2)$ then $n_1 \neq n_2$. It means that the Cloud administrator can group servers in a logical way, namely *sites*, for her needs. Usually, sites should be used to group servers geographically close to each other, however it is also possible to associate servers to a same site for various reasons, as for example hardware type. The set of all sites is denoted $\mathcal{S}$.

Tag: the tag concept is close to the site one. It associates one or multiple string tags to a given node, thus building subsets of servers $T \subset (\mathcal{N} \cup \mathcal{E})$. Two main differences can be noticed though compared to sites. First, any node $n \in (\mathcal{N} \cup \mathcal{E})$ can be associated to one or multiple tags, while the same node can only be linked to a single site. Second, tags are not used into CSP but used as filters, as it will be described in the next section.

B. SkyPlace view

The SkyPlace view (SP) offers constraints which are specific to hybrid Clouds. It takes advantages of the three new infrastructure concepts *extern*, *site* and *tag*, and proposes four placement constraints and one filter mechanism described below.

OnSite: If a virtual machine is associated to an *OnSite* constraint, it has to be hosted by a node included in the given site. More formally, for a virtual machine $v_j \in \mathcal{V}$ and a site $S \in \mathcal{S}$ such that the constraint $OnSite(S, v_j)$ is applied, the following property is verified:

$$h_{ij} = 1 \Rightarrow n_i \in S \quad (5)$$

OffSite: On the other hand, if a virtual machine is associated to an *OffSite* constraint, it must not be hosted by a node included in the given site. More formally, for a virtual machine $v_j \in \mathcal{V}$ and a site $S \in \mathcal{S}$ such that the constraint $OffSite(S, v_j)$ is applied, the following property is verified:

$$h_{ij} = 1 \Rightarrow n_i \notin S \quad (6)$$

Near: A *Near* constraint is applied to a subset of virtual machines $\mathcal{V}_{near} \subset \mathcal{V}$ such that these virtual machines must be hosted onto the same site $S \in \mathcal{S}$. More formally:

$$\left| \bigcup_{\substack{n_i \in \mathcal{N} \\ h_{ij}=1 \\ v_j \in \mathcal{V}_{near}}} site(n_i) \right| = 1 \quad (7)$$

Far: The *Far* constraint is applied to a subset of virtual machines $\mathcal{V}_{far} \subset \mathcal{V}$ such that these virtual machines must be hosted onto different sites of $\mathcal{S}$. More formally:

$$\left| \bigcup_{\substack{n_i \in \mathcal{N} \\ h_{ij}=1 \\ v_j \in \mathcal{V}_{far}}} site(n_i) \right| = |\mathcal{V}_{far}| \quad (8)$$

One can note that the *Far* constraint is close to the *spread* constraint defined in Equation (2). However this constraint is applied to sites instead of servers. Thus, *Far* and *Spread* constraints are applied at different levels.

Tag filter: As already mentioned, *tags* are not used into CSP. Filters are associated to tags to reduce the overall searching space of nodes to consider for each virtual machine. Thus, before applying the CSP solver, every virtual machine is associated to a subset of possible nodes included in $T \subset (\mathcal{N} \cup \mathcal{E})$, all associated to the given tag. As a result, a tag is a filter solution, while sites are used to express complex constraints solved by the Choco CSP solver.

V. Evaluation

In this section we evaluate the use of OptiPlace for a real case study. First, we describe in details the use case study application, namely TrustyDrive. Second, we present two different placement problems which could be defined to deploy TrustyDrive onto an hybrid Cloud. Being able to describe all TrustyDrive constraints by using the new SP view shows a good level of expressivity. Finally, we discuss about performances of the new SP view.

A. TrustyDrive

As previously introduced in Section II, at least four classes of applications can benefit from Cloud federation. Among them are, the class of applications that need replication of services for reliability, and the class of applications that want to place data or computations within a specific country, region or site for security and data privacy reasons.

In this paper we study an application relevant to both these classes, namely *TrustyDrive* [13]. TrustyDrive is a distributed storage system that uses many Cloud providers instead of a single one to store user data. TrustyDrive guarantees data privacy as well as a reliable storage. The idea behind TrustyDrive is to divide sensitive documents into multiple chunks and to place these encrypted chunks onto different FTP servers themselves placed onto different Cloud providers. As a result, TrustyDrive guarantees that none of the Cloud providers has the possibility to rebuild an entire protected document. Moreover, to be more reliable TrustyDrive can be combined with erasure code algorithms that provides RAID features at the Cloud storage level. By using such erasure algorithm, intelligent duplication of information are computed to be able to restore the document even if k chunks are lost.

OptiPlace is used to solve the placement problem associated to the deployment of TrustyDrive onto Cloud providers. Actually, TrustyDrive is a distributed software composed of one dispatcher (or master) and n FTP servers (or clients) that have to be placed onto an hybrid Cloud infrastructure by respecting some constraints.

B. Basic use-case

The use case of Figure 1 is used as a basis in our evaluation. This use case represents a simple deployment of TrustyDrive. It also represents the minimum set of constraints to guarantee the good behavior of TrustyDrive. Actually, while the dispatcher and the web interface need to be hosted onto the local private Cloud (for security reasons), each FTP server has to be hosted onto a distant external host. Moreover, to guarantee data privacy and security assumed by TrustyDrive, each FTP server must be hosted onto a different site.

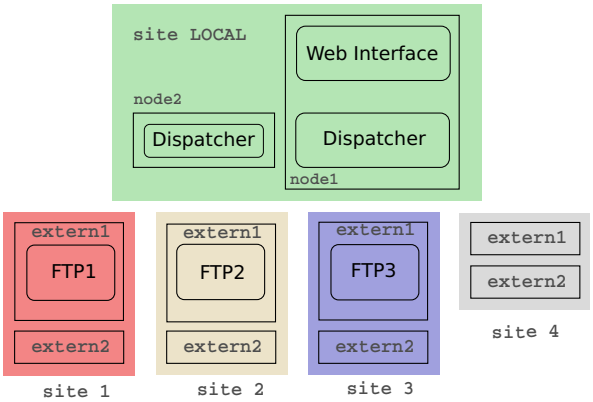

Figure 1: Basic experimental setup. Deployment of a basic configuration of TrustyDrive with one WI, two Dispatchers (for fault tolerance reasons) and three FTP servers.

The considered infrastructure for this basic use case (depicted in Figure 1) is defined as follow:

- five sites are declared $\{local, S_1, S_2, S_3, S_4\} \subset \mathcal{S}$;
- the site $local$ represents the private Cloud and is composed of two nodes $\{n_1, n_2\} \subset \mathcal{N}$, both of them with a capacity (memory) of 2Gb;
- each site S_1 to S_4 is composed of two types of external nodes $e_1, e_2 \in \mathcal{E}$. both with a capacity of 1Gb.

The set of virtual machines have to be defined and placed according to a set of constraints as follows:

- one virtual machine is used for the web interface $v_{wi} \in \mathcal{V}$ and needs a memory capacity of 1Gb;
- two virtual machines are built for the dispatcher of TrustyDrive $v_{d1}, v_{d2} \in \mathcal{V}$, both of them need a capacity of 5Mb. One of the two dispatcher is not used except if the first one fall down, thus it guarantees that the dispatcher will continue to work. To ensure this reliability, the two dispatchers have to be placed onto different servers, thus we define the constraint $Spread(v_{d1}, v_{d2})$ of HA;
- to ensure that the dispatcher and the web interface are hosted onto a local site (for security reasons) an OnSite constraint is applied as follows: $OnSite(local, v_{wi}, v_{d1}, v_{d2})$;
- one virtual machine is created for each FTP server v_{ftp1}, v_{ftp2} and $v_{ftp3} \in \mathcal{V}$, each of them need a capacity of 1Gb;
- FTP servers must not be hosted by $local$, following the constraint $OffSite(local, v_{ftp_1}, v_{ftp_2}, v_{ftp_3})$;

- to guarantee data privacy of TrustyDrive v_{ftp1}, v_{ftp2} and v_{ftp3} must by placed onto different external Cloud providers. Thus, the following constraint is defined: $Far(v_{ftp1}, v_{ftp2}, v_{ftp3})$;

As a result, this use case is composed of five sites, five nodes, eight external nodes, six virtual machines and four constraints. By defining the above constraints, the placement solution represented in Figure 1 is computed by OptiPlace. This use case show the usefulness of the new SkyPlace view of OptiPlace. Without the four constraints $Spread$, $OnSite$, $OffSite$ and Far, the expected deployment of TrustyDrive could not be guaranteed automatically and would have to be solved manually. By using the view a simple configuration file could be given by an administrator to precise constraints.

For example, if the constraint $Far(v_{ftp1}, v_{ftp2}, v_{ftp3})$ was not given, the placement solution computed by OptiPlace would have been to place all FTP virtual machines onto one extern type of site 1, because the number of ocurrence of an external node is unlimited.

C. Advanced use-case

To illustrate more clearly the interest of OptiPlace, we also would like to consider a more complex example represented in Figure 2. In this example, all FTP servers are duplicated to ensure fault tolerance of TrustyDrive.

We consider that the underlying infrastructure is defined as follows:

- three different tags are declared: *FRANCE*, *USA* and *ENGLAND*. These tags represents three different countries;
- eight sites are declared $\{local, S_i, i \in [\![1, 8]\!]\} \subset \mathcal{S}$;
- the site $local$ represents the private Cloud and is composed of two nodes $\{n_1, n_2\} \subset \mathcal{N}$, both of them with a capacity (memory) of 2Gb;
- each site S_1 to S_8 is composed of two external nodes $e_1, e_2 \in \mathcal{E}$ both with a capacity of 1Gb;
- Nodes of sites S_1 and S_4 are tagged as "USA", nodes of sites S_0, S_2, S_3 and s_7 are tagged as "FRANCE", and finally nodes of sites S_5 and S_6 are tagged as "ENGLAND".

The set of virtual machines and the set of constraints are defined as follows:

- the web interface virtual machine $v_{wi} \in \mathcal{V}$, as well as the dispatchers virtual machines $v_{d1}, v_{d2} \in \mathcal{V}$ are declared the same way than in the previous example, and the constraints $OnSite(local, v_{wi}, v_{d1}, v_{d2})$ and $Spread(v_{d1}, v_{d2})$ are also applied;
- one virtual machine is created for each FTP server and its associated replica v_{ftp1}, $v_{ftp1'}$, v_{ftp2}, $v_{ftp2'}$, v_{ftp3} and $v_{ftp3'} \in \mathcal{V}$;
- FTP servers must not be hosted by $local$ site, following the constraint $OffSite(local, v_{ftp_1}, v_{ftp_2}, v_{ftp_3})$;
- the filter $filter("FRANCE")$ is applied to all FTP virtual machines to exclude from $\mathcal{N} \cup \mathcal{E}$ external nodes associated to tags USA and $ENGLAND$;
- to guarantee data privacy of TrustyDrive v_{ftp1}, v_{ftp2} and v_{ftp3} must by placed onto different Cloud providers. Thus, the following constraint is defined: $Far(v_{ftp1}, v_{ftp2}, v_{ftp3})$;
- in this example, we also want to place the replica on the same site than the FTP server it replicates. For this reason, three constraints are added: $Near(v_{ftp1}, v_{ftp1'})$, $Near(v_{ftp2}, v_{ftp2'})$ and $Near(v_{ftp3}, v_{ftp3'})$;
- finally, we add three constraints such that the replica is placed onto a different server than its parent FTP VMs: $Spread(v_{ftp1}, v_{ftp1'})$, $Spread(v_{ftp2}, v_{ftp2'})$ and $Spread(v_{ftp3}, v_{ftp3'})$.

Compared to the basic example, this advanced use-case uses the new constraint $Near$, an additional use of $Spread$ and the $tags$. Thus, this use-case uses every constraint proposed by the new SkyPlace view of OptiPlace. Once again without those constraints it would not be possible to get the expected placement solution depicted in Figure 2. For example, without both $Spread$ and $Near$ constraints, the Far constraint would still guarantee the good placement of v_{ftp1}, v_{ftp2} and v_{ftp3} (as for the basic example), however, because of unlimited external nodes, all replicas $v_{ftp1'}$, $v_{ftp2'}$ and $v_{ftp3'}$ would be placed onto e_1 of S_0.

D. Cost minimization

In this paper we do not bring as a contribution a heuristic to efficiently reach cost optimization. However, and as detailed in Section III-B, OptiPlace is already implemented with this mechanism with the HC view. To illustrate the interest of this mechanism onto the new constraints proposed in this paper, we have performed the following evaluations. First, in the basic example of Figure 1, we have added a default cost 1 to each site, and the second site has a specific cost of 100. As a result, the fourth site is preferred to the second one to minimize the cost of the placement solution.

Second, in the advanced example, we have added a default cost 1 to each site, the second site has a specific cost of 200, and the third site has a specific cost of 100. Thus by following this inequation, $cost(site2) > cost(site3) > cost(site7)$, the seventh site is preferred to both the second and third sites, and the third site is preferred to the second one. As a result, the second site stays empty.

Both those examples show that the cost minimization of the HC view could be used within the placement problem to improve the placement choice according to the cost.

E. Performances

Both basic and advanced use-cases has been performed 33 times, the execution times has been registered and the median has been computed. Results are represented in Table I.

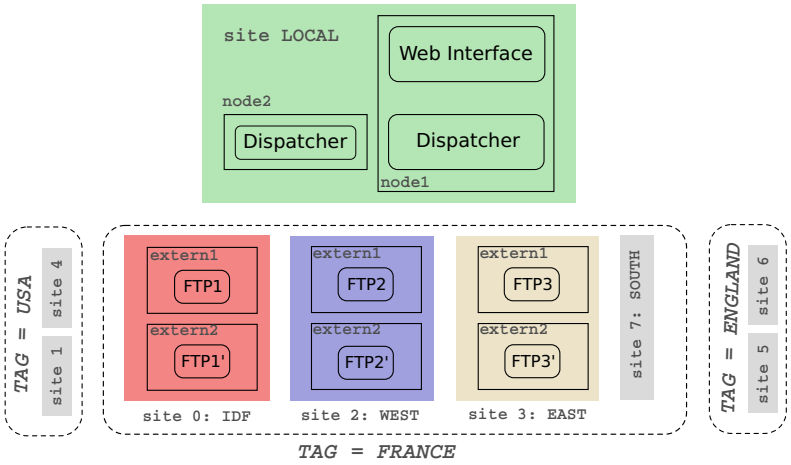

Figure 2: Advanced experimental setup. Deployment of a complex configuration of TrustyDrive with one WI, two Dispatchers (for fault tolerance reasons) and three FTP servers. Each FTP server is duplicated (for safety reasons) close to the initial one (on the same site for performance reasons in case of fault) but onto a different server.

	time (ms)
basic	4.6
advanced	6.07

Table I: Execution times for basic and advanced use-cases of TrustyDrive.

Results show that the execution time to solve both those use-cases are very small. One can note, as expected, that the advanced example is longer to solve. Actually the number of variables and constraints are more important in the advanced example.

Same experiments has been conducted by adding the Host Cost view. Results are represented in Table II. In addition to execution time, the number of backtracks is represented. A backtrack happens when the constraint solver go back in the solution tree to explore another branch.

	time (ms)	backtracks
basic	6.6	19
advanced	14.6	619

Table II: Execution times and number of bactracks for basic and advanced use-cases of TrustyDrive, when Host Cost optimization is enabled.

Results show that the execution time to solve these use-cases are still very small even if the cost is minimized. However, one can note that the advanced use-case is 150% longer with cost minimization than without. As already described, without the Host Cost view the problem to solve is to find one valid solution of the placement problem. On the other hand, when the Host Cost is activated, a function has to be minimized which means that all solutions should be processed to find the best one. For this reason, activating the Host Cost view is more expensive and could lead to very long time processing. Heuristics could be proposed to improve this mechanism and will be the subject of future work.

To illustrate the performance behavior of OptiPlace, we have increased the number of virtual machines to place (FTP VMs) while the number of nodes, externs and sites stay constant (100 sites composed of two extern nodes each). Results are shown in Figure 3. As expected the execution time increase linearly with the number of VMs. Two reasons are responsible for that: first, the number of variables in the CSP increases, which results in a bigger CSP-tree to entirely explore when HC is activated; second, the number of solutions decreases (as the infrastructure stays the same), thus it is longer to find one good solution, even when HC is desactivated.

One can note that the execution time to place 100 virtual machines onto an infrastcuture composed of 100 sites, each of them containing 2 external nodes (top-right point of Figure 3), does not exceed 60 ms which is a short execution time to deploy TrustyDrive onto an hybrid Cloud. Actually, TrustyDrive is an ever-running application which will occasionnally need reconfigurations. Thus, spending 60 ms for its deployment seems satisfactory.

Finally, one can note that activating the HC view globally results in a biger execution time. Actually, as the exact solution is found by HC, all valid solutions should be found before the cost minimization can be chosen.

VI. Conclusion

In this paper has been presented the new SkyPlace view of the OptiPlace framework. This view is responsible for the integration of federation and hybrid Cloud considerations

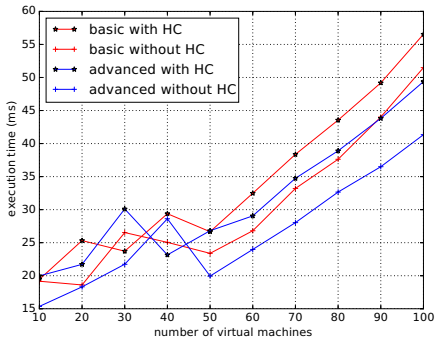

Figure 3: Execution times when the number of VMs increase. The infrastructure size stays constant.

into OptiPlace, thus solving new types of placement problems. After an introduction on the constraint solver Choco and the OptiPlace framework, the new set of variables and constraints introduced into SkyPlace has been presented. The application TrustyDrive, *i.e.*, a distributed reliable and privacy-aware storage system, has been used to build two use-cases. Those use-cases have shown that the constraints proposed by SkyPlace are adapted to hybrid Clouds and federation of Clouds. Moreover, a performance study has been presented to show that the behavior of such placement problem regarding the size of the problem is very complex. It has been shown, though, that the execution times of SkyPlace on both use-cases are satisfactory regarding the nature of the application.

As already evocated in the paper, proposing heuristics to improve the execution time of the HostCost view will be the subject of future work. If the interest of such heuristics is not obvious for the TrustyDrive use-case, highly dynamic applications such as smart-* applications and real-time applications (virtual reality etc.) could be very sensitive to the execution time needed to solve a placement problem. Moreover, future generations of utility computing such as Edge or Fog computing could lead to very large infrastructures for which SkyPlace should propose better scaling results regarding the number of virtual machines to manage.

Acknowledgment

References

[1] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud Federation," in *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*. IARIA, Sep. 2011, Inproceedings. [Online]. Available: http://www.aifb.kit.edu/images/0/02/Cloud_Federation.pdf

[2] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures," *Computer*, vol. 45, no. 12, pp. 65–72, Dec 2012.

[3] F. Hermenier and S. Demassey, "BtrPlace: Flexible VM Management in Data Centers," in *Conference on Optimization & Practices in Industry, PGMO-COPI'14*, Paris, France, Oct. 2014. [Online]. Available: https://hal.inria.fr/hal-01095958

[4] P. Silva, C. Pérez, and F. Desprez, "Efficient Heuristics for Placing Large-Scale Distributed Applications on Multiple Clouds," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'16)*, Cartagena, Colombia, May 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01301382

[5] G. L. Louët and J. Menaud, "Optiplace: Designing cloud management with flexible power models through constraint programming," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013*, 2013, pp. 211–218.

[6] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, Sept 2009.

[7] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *Journal of Parallel and Distributed Computing*, vol. 72, no. 10, pp. 1318 – 1331, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731512001517

[8] R. V. den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 228–235.

[9] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Gener. Comput. Syst.*, vol. 28, no. 2, pp. 358–367, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.07.003

[10] D. P. Chandu, "A parallel genetic algorithm for three dimensional bin packing with heterogeneous bins," *CoRR*, vol. abs/1411.4565, 2014. [Online]. Available: http://arxiv.org/abs/1411.4565

[11] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Documentation*, TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., 2016. [Online]. Available: http://www.choco-solver.org

[12] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. L. Lawall, "Entropy: a Consolidation Manager for Clusters," in *VEE 2009 - 5th International Conference on Virtual Execution Environments*. Washington, DC, United States: ACM, Mar. 2009, pp. 41–50. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01102354

[13] R. Pottier and J.-M. Menaud, "TrustyDrive: a Multi-Cloud Storage Service that Protects Your Privacy," in *IEEE 9th International Conference on Cloud Computing*, ser. International Conference on Cloud Computing, San Francisco, United States, Jun. 2016. [Online]. Available: https://hal.inria.fr/hal-01322638