Lagab Yanis
Reffay Thomas
Vialatte Sylvain

M1 IMIS TER
2022

Programmation par contraintes pour un courtier Cloud orienté sécurité

Rapport de Travaux d'études et de recherche et Technique de communication

1/ Introduction

L'objectif de ce sujet de TER est de proposer un courtier cloud codé en programmation par contrainte et qui dispose des mêmes caractéristiques que celui proposé par Salwa Souaf dans sa thèse "*Formal methods meet security in a cost aware cloud brokerage solution*".

Il nous a fallu dans un premier temps bien comprendre la solution proposée par Souaf dans sa thèse, et donc lire attentivement cette dernière pour en extraire les caractéristiques du courtier cloud.

Nous avons ensuite fait un état de l'art sur les méthodes de programmation par contraintes appliquées au cas des courtiers cloud.

Enfin, grâce à ces informations, nous avons pu nous intéresser à l'implémentation de notre courtier.

2/ Fonctionnement d’un courtier cloud

Un courtier cloud est une application qui agit comme un intermédiaire entre les vendeurs de services cloud (Amazon, Microsoft,...) et les clients qui souhaitent louer des ressources cloud.

Un client se présente avec des demandes (il a besoin de faire fonctionner certaines applications, qui ont des exigences fonctionnelles, et il -le client- peut avoir en plus des exigences non fonctionnelles, souvent liées à la sécurité) et le courtier lui présente une combinaison de vms qui permet de répondre à ses exigences.

Cela ressemble à un problème de placement vu sous cet angle, mais pour les clients, il est aussi important de réduire les coûts au maximum et donc cela devient un problème d’optimisation.

C’est un problème NP-difficile et plusieurs approches sont étudiées pour le résoudre dans un temps le plus court possible. Par exemple la programmation linéaire ou encore comme dans notre cas la programmation par contraintes.

Le courtier cloud présenté par Salwa SOUAF a pour mission de résoudre un problème de placement de VMs sur des serveurs physiques en respectant un certain nombre d’exigences, en particulier en prenant en compte des exigences de sécurité.

Le courtier fonctionne par étape. Dans un premier temps, les fournisseurs cloud lui attribuent des ressources qu’il pourra par la suite utiliser pour répondre aux attentes des clients.

Ensuite, quand un client effectue une demande, le courtier commence par vérifier la cohérence des exigences fonctionnelles (la quantité et la description des ressources) et des exigences non fonctionnelles (propriétés de sécurité). En effet, rien n’indique au préalable que la demande du client est réalisable, même en considérant qu’on a accès à toutes les vms imaginables, sa demande pouvant être incohérente.

Si les exigences du client ne sont pas cohérentes, alors le courtier prévient le client et lui présente un contre-exemple afin de mettre en évidence les éventuelles incohérences.

Une fois que les exigences du client sont cohérentes, le courtier s’applique à trouver une disposition valide et fait ensuite une proposition de déploiement au client. Le client peut confirmer la proposition, auquel cas, il est par la suite guidé dans le déploiement.

3/ Modélisation du courtier cloud de Salwa Souaf

Le courtier cloud de Salwa Souaf repose sur trois principales parties différentes, la relation entre les courtières et les fournisseurs de cloud, la relation entre le courtier et le client et enfin la recherche de solution pour le problème de placement en lui-même.

Mais avant d'expliquer ces 3 points, nous allons d'abord décrire les différentes modélisations.

Le courtier doit préalablement réserver auprès des fournisseurs d'accès cloud un certain nombre de produits, de façon à pouvoir par la suite les proposer aux clients. Pour représenter ces services des "VMs" sont utilisées. Concrètement c'est une liste de caractéristiques qui définissent le produit du courtier.
Cette liste se compose

- d'un OS (ex: Windows)
- d'un nombre de coeur virtuel pour le processeur (une mesure de la puissance de calcul) (ex: 8)
- d'une quantité de mémoire RAM (ex: 16 Go)
- d'une localisation géographique (ex: France)
- d'un coût (ex: 200€/an)
- d'un fournisseur (ex: Amazon)

Le courtier dispose ainsi d'une liste de produits qu'il peut utiliser pour satisfaire les demandes du client.

De son côté, le client modélise ses exigences par une liste de "composants". Un composant est lui aussi une liste de caractéristiques qui sont :

- d'un OS (ex: Windows)
- d'un nombre de coeur virtuel pour le processeur (une mesure de la puissance de calcul) (ex: 8)
- d'une quantité de mémoire RAM (ex: 16 Go)
- d'une localisation géographique (ex: France)

Le but du courtier va être d'attribuer à chaque composant une vm sur laquelle il pourra s'exécuter.
Le client peut également préciser des relations entre ses composants qui seront des contraintes supplémentaires pour le courtier:

➔ flow(x,y) qui indique que les deux composants x et y doivent pouvoir disposer d'un canal de communication

- ➔ alone(x) qui indique que le composant doit être le seul composant placé sur la vm sur laquelle il sera placé
- ➔ concurrence(x,y) qui indique que les deux composants x et y ne doivent pas disposer d'un canal de communication direct ou indirect

À l'aide de cette modélisation du problème et de règles de programmation linéaire, on peut résoudre le problème de placement.

Le courtier réserve à l'avance une certaine quantité de ressources et avec des caractéristiques différentes, de différents fournisseurs. L'approvisionnement pourrait se faire en tenant compte du nombre de clients et de ressources utilisées ou non utilisées, sur une période donnée. Une période appropriée sera sélectionnée et une estimation des ressources sera effectuée avant le début de chaque période. Nous supposerons que le courtier disposera toujours des ressources nécessaires pour répondre à la demande du client.

Le client spécifie sa demande au courtier en précisant les exigences fonctionnelles et non fonctionnelles. Puis le courtier vérifie la cohérence de la requête et si celle-ci ne contient pas des exigences contradictoires.

Puis, le courtier recherche un placement qui répond aux exigences fonctionnelles du client tout en réduisant au minimum le coût d'installation. Une fois que le client a approuvé la stratégie de placement retournée, le courtier s'occupe alors du déploiement et des configurations de réseau appropriées pour satisfaire les exigences de sécurité du client.

4/ État de l'art

Différents courtiers cloud ont été créés auparavant, certains utilisent la programmation linéaire, d'autres la programmation par contrainte. Nous avons étudié surtout 3 courtiers cloud :

- Le courtier cloud BtrPlace, nous nous sommes basés sur une publication datant de 2013 et écrite par Fabien Hermenier, Julia Lawall et Gilles Muller.
- Le courtier OptiPlace qui est une amélioration de BtrPlace et est cité dans une publication de 2017 écrite par Hélène Coullon, Guillaume Le Louet et Jean-Marc Menaud
- Les courtiers décrits dans la thèse de Salwa Souaf, ceux-ci sont assez similaires et le fonctionnement de la dernière version de ces courtiers a été décrite dans la partie d'avant (Partie 3).

Papier de 2013 : "BtrPlace: A Flexible Consolidation Manager for Highly Available Applications"

BtrPlace est un gestionnaire de consolidation flexible qui peut être configuré dynamiquement avec des contraintes de placement, il s'appuie sur la programmation par contrainte pour modéliser et résoudre les reconfigurations problem, le core reconfiguration problème c'est de calculer une configuration viable en choisit d'abord pour chaque VM un serveur d'hébergement qui répond aux besoins des VM, puis planifier les actions qui convertiront la configuration actuelle en la configuration choisie, Le calcul d'une solution pour la reconfiguration problème peut prendre du temps de consommation pour les grands data center donc l'optimisation du processus de résolution BtrPlace utilise 3 stratégies la simplification du full Reconfiguration Problem, heuristics guiding the CP solver pour de la reconfiguration plans rapide et le Partitionnement.

Papier de 2017:

Dans "*Virtual Machine Placement for Hybrid Cloud using Constraint Programming*", OptiPlace est utilisé et sert de base de travail. L'objectif étant de proposer des améliorations à OptiPlace dans le cas spécifique des clouds hybrides. C'est-à-dire les situations où un client souhaite disposer d'un cloud privé qui lui appartient et dans le même temps, louer des ressources auprès de cloud providers.

De base, OptiPlace a été conçu pour répondre à un problème de reconfiguration capable de gérer un seul fournisseur de cloud à la fois, interdisant la possibilité d'avoir une fédération ou une solution de cloud hybride.

Les contributions de ce papier :

- un modèle générique d'infrastructure pour pouvoir gérer les clouds privés et publics en même temps
- de nouvelles contraintes pour le contexte des clouds hybrides
- une évaluation des nouvelles contraintes proposées

Un ensemble d'*external nodes* est noté ε. Un external node est équivalent à un internal node sauf qu'il est disponible en quantité infinie. Il n'est pas non plus associé à des composants hardware. Le courtier pourra privilégier les nœuds externes ou internes en fonction des coûts ou des limites d'alimentation.

Il y a également *Site* : un *site* est un sous-ensemble de serveurs, chaque *node* n'est associé qu'à un seul site. Le concept d'un *site* est de pouvoir regrouper des serveurs ensemble en fonction de caractéristiques qu'ils partagent. Que cela soit matériel ou géographique. L'ensemble de tous les *site* est noté S.

Et enfin *Tag* : chaque *node* peut être associé à un ou plusieurs *tag*, ce qui crée des sous-ensembles. Très proche de *site*, il y a tout de même deux différences majeures : un *node* peut avoir plusieurs *tags* et les *tags* ne sont pas utilisés dans le CSP.

La *SkyPlaceView* propose ensuite de nouvelles contraintes qui utilisent les nouveaux concepts d'infrastructure cités au-dessus :

- *Onsite* : Une certaine machine doit appartenir à un certain site. Formellement pour une machine $v_j \in$ V et un site s $\in$ S, la propriété *OnSite*(S,v_j) signifie : $h_{ij} = 1 \Rightarrow n_i \in S$
- *OffSite*: Exactement l'inverse la machine v_j ne doit pas être hébergée sur le site s. Plus formellement pour une machine $v_j \in$ V et un site s $\in$ S, la propriété *OffSite*(S,v_j) signifie : $h_{ij} = 1 \Rightarrow n_i \notin S$
- *Near :* indique qu'un ensemble de machines virtuelles doivent être hébergées sur le même *site*
- *Far* : indique qu'un ensemble de machines virtuelles doivent toutes être hébergées sur différents *sites*

5/ Définition d’un modèle

La programmation par contrainte se base essentiellement sur la définition du problème et sur la modélisation de celui-ci. Nous avons dû réaliser un CSP (problème de satisfaction de contraintes), il est défini par des variables, leurs domaines et des contraintes.

Avant tout, nous avons besoins de poser des données :

- NComp : le nombre de composants
- NMachine : le nombre de machines
- NLocalisation : le nombre de localisations différentes disponibles
- NOs : le nombre d’Os différents disponibles

Avec ceci nous avons construit différents tableaux afin de représenter les données disponibles :

- Localisations le tableau des localisations : tableaux de string de taille NLocalisation
- OS le tableau des systèmes d’exploitations : tableaux de string de taille NOs

Dans le but de modéliser les composants, l’on a construit un tableau d’entier de taille NComp pour chaque caractéristique technique :

- CompOs : pour les systèmes d’exploitation des composants
- CompMem: pour les capacités de mémoires des composants
- CompProc: pour le nombre de processeurs des composants
- CompLoc: pour les localisations des composants

Idem pour les machines, l’on a un tableau d’entier de taille NMachine par caractéristique :

- VmOs : pour les systèmes d’exploitation des machines
- VmMem: pour les capacités de mémoires des machines
- VmProc : pour le nombre de processeurs des machines
- VmLoc : pour la localisation des machines
- VmCout : pour le coût de location des machines
- VmForum : pour le fournisseur des machines

Grâce à ces données nous avons pu en déduire 2 variables, nous avons défini les tableaux suivant :

- le premier est un tableau d’entier représentant sur quelle machine est hébergé chaque composant
- le second est un aussi tableau d’entier comptant le nombre de composants sur une machine

Leurs domaines sont respectivement de 0 à NComp et de 0 à NMachine.

Grâce à tout cela, on peut maintenant construire des contraintes afin de résoudre notre CSP. On a défini 5 contraintes dans le but de placer correctement les composants sur les VM disponibles.

La localisation de chaque composant doit correspondre à celui de la vm sur laquelle il est hébergé.

$$\forall i \; allant \; de \; 1 \; \text{à} \; N, \; Comp_{Loc}[i] = VM_{Loc}[X[i]]$$

La seconde est sensiblement la même que la première, mais pour l'OS.

L'os de chaque composant doit correspondre à celui de la vm sur laquelle il est hébergé.

$$\forall i \; allant \; de \; 1 \; \text{à} \; N, \; Comp_{OS}[i] = VM_{OS}[X[i]]$$

La troisième sert à faire en sorte que le nombre de processeurs d'une machine est supérieur ou égale à la somme des processeurs des composants hébergée sur cette machine. Même chose pour la quatrième, mais avec la capacité de mémoire.

La somme du nombre de cœurs de processeurs requise par les composants hébergés sur une machine v ne doit pas excéder le nombre de cœurs de processeur de la machine v.

$$\forall v \; allant \; de \; 1 \; \text{à} \; M, \; \sum_{i=1}^{n} ((X[i] = v) * Comp_{PROC}[i]) \leq VM_{PROC}[v]$$

La somme de la mémoire requise par les composants hébergés sur une machine v ne doit pas excéder la mémoire disponible sur la machine v.

$$\forall v \; allant \; de \; 1 \; \text{à} \; M, \; \sum_{i=1}^{n} ((X[i] = v) * Comp_{MEM}[i]) \leq VM_{MEM}[v]$$

Il y a également une fonction de coût que l'on souhaite minimiser. Elle est définie comme suit:

Le coût total de la configuration correspond à la somme du coût de toutes les machines v utilisées pour héberger des composants.

$$min(\sum_{v=1}^{m} (Y[v] \geq 0) * VM_{Cout}[V])$$

L'ensemble $Comp_{Alone}$ est un ensemble qui rassemble des composants qui ne doivent être les seuls de la configuration à être hébergés sur leur machine.

$$Comp_{Alone} : \forall i,\ i \in Comp_{Alone},\ \forall j,\ j \in [1, N],\ i \neq j \Rightarrow X[i] \neq X[j]$$

6/ Implémentation

Pour la partie implémentation nous avons tout d'abord décider de commencer par faire un jeu de données afin de pouvoir tester notre futur programme et de mieux comprendre pourquoi le modèle ne fonctionnerait pas. l'architecture et le même que le modèle présenté dans la partie modélisation.

```
%Rapport localisation et os / entier
Localisations = {"Suisse","France","Etats-Unis","Italie"};
OS = {"Windows","Linux"};

%Spec Composant
%Array des OS array[1..NComp] of string
CompOs = [1,1,1,2,2];
%Array des memoires array[1..Ncomp] of int
CompMem = [4,8,4,4,8];
%Array des processeurs array[1..Ncomp] of int
CompProc = [2,2,2,4,2];
%Array des localisations array[1..NComp] of string
CompLoc = [2,2,2,1,1];
```

Jeu de données MiniZinc : Représentation des composants

Par exemple ici nous avons un jeu de données avec 5 composants à placer et 10 machines disponibles avec 4 localisations différentes et 2 systèmes d'exploitation différents. Ci-dessus vous pouvez voir les caractéristiques des différents composants disponibles. Pour les tableaux CompOS et CompLoc, les valeurs correspondent à un indice des tableaux OS et Localisation. Le Composant 1 est représenté par toutes les valeurs d'indice 0, le composant 2 par les valeurs d'indice 2,..

```
%Spec Machine
%Array des OS array[1..NMachine] of string
VMOs = [2,1,1,1,2,2,1,1,2,2];
%Array des memoires array[1..NMachine] of int
VMMem = [16,8,8,8,8,8,8,8,16,32];
%Array des processeurs array[1..NMachine] of int
VMProc = [8,4,4,4,8,2,2,8,8,4];
%Array des localisations array[1..NMachine] of string
VMLoc = [1,2,2,2,1,1,3,3,3,4];
%Array dess cout array[1..NMachine] of int
VMCout = [100,50,60,30,40,78,32,40,100,120];
%Array des fournisseurs array[1..NMachine] of string
VMFourn = ["Amazon","Microsoft","Amazon","Microsoft","Amazon","Microsoft","Amazon","Microsoft","Amazon","Microsoft"];
```

Jeu de données MiniZinc : Représentation des machines

Pour les machines cela fonctionne de la même manière que les composants. Le résultat qui est attendu sur ce jeu de données est que les composants 1, 2 et 3 soient répartis sur les composants 2, 3 et 4 (qu'importe qui est hébergé ou), les composant 4 et 5 doivent, quant à eux, être hébergés sur la machine 1.

Nous avons essayé deux implémentations différentes, une en minizinc et l'autre en python en utilisant la librairie cpmpy.

Nous avons commencé sur conseil de notre enseignante encadrant par essayer d'implémenter notre modèle en python à l'aide de la librairie cpmpy. Elle permet de faire de la programmation par contraintes sur python en utilisant notamment le calcul vectoriel pour raccourcir et éclaircir la syntaxe.

Nous avons commencé par bloquer sur un problème de type. En effet, les variables ne sont pas du même type que les données et ne sont pas non plus un type de base. Nous pensions avoir résolu le problème avec .value() sur une intvar.
La rédaction de la contrainte "count" nous a aussi posé problème. Elle ne semble pas intégrée à la librairie.
Après quelques recherches dans la documentation, nous sommes parvenus à créer un solveur sans erreur.

Mais malheureusement, quand nous essayons de lui faire résoudre le modèle et donc trouver quelles sont les affectations de valeurs aux différentes variables possibles, on obtient une erreur que nous n'arrivons pas à comprendre. Et quand nous essayons de nous renseigner sur Internet sur cette erreur, nous ne trouvons aucun résultat.

Voici l'erreur :

```
AttributeError: 'numpy.bool_' object has no attribute 'is_bool'
```

Rapport d'erreur de python

Elle est indiquée comme apparaissant à la ligne model.solve() mais sans plus de détails.

Nous avons réussi à déterminer que l'erreur provient des contraintes sur l'OS et la localisation, mais nous ne savons pas comment la résoudre.
Elle a l'air d'être en lien avec les .value() que nous faisons sur les variables mais si nous ne les faisons pas, nous ne pouvons utiliser cette syntaxe car les tableaux numpy_array n'attend que des entiers dans les opérateurs d'accès et pas d'intvar.

Aussi, le type numpy_array qui est le type de tableau utilisé par cpmpy a l'avantage d'autoriser le calcul vectoriel mais n'est pas aussi simple à manipuler qu'un tableau classique.

Notre blocage nous a amené à essayer d'implémenter notre modèle sur minizinc. En effet, nous nous sommes déjà servis de minizinc pour notre cours sur la programmation par contraintes, donc nous disposons d'exemples et de cours et surtout d'un peu plus d'expérience que sur la librairie cpmpy.

Mais nous n'arrivons pas à exprimer toutes les contraintes sans erreur. La contrainte de count en particulier nous pose des difficultés. En effet aucune des signatures de la fonction en minizinc ne correspond à l'usage que nous cherchons à en faire.

```
/home/etud/o2173830/Documents/TER/ter-broker-cloud-et-pcc/Modele_MiniZinc/modele1.mzn:
46.42-51:
MiniZinc: type error: no function or predicate with this signature found:
`count(array[int] of var int,int)'
Process finished with non-zero exit code 1
Finished in 53msec
Compiling modele1.mzn
/home/etud/o2173830/Documents/TER/ter-broker-cloud-et-pcc/Modele_MiniZinc/modele1.mzn:
46.37-52:
MiniZinc: type error: no function or predicate with this signature found:
`count(array[int] of var int,int,var int)'
Process finished with non-zero exit code 1
Finished in 52msec
```

Rapport d'erreur de MiniZinc

Nous avons dû faire une petite modification sur les données de Localisation et d'OS. En effet, en minizinc, on ne peut pas utiliser une variable pour accéder à un certain indice des tableaux de chaînes de caractère.

Sachant qu'il nous restait peu de temps, nous avons voulu essayer d'avoir un modèle qui fonctionne et donc nous avons mis en place quelque chose d'assez peu imaginable en pratique : une liste de toutes les localisations possibles et une liste de tous les OS possibles.

Ensuite pour CompLoc et CompOs et VMLoc et VMOs au lieu d'être des tableaux de chaînes de caractère, ce sont des tableaux d'entiers, l'entier étant l'indice de la localisation dans le tableau des localisations.

De cette façon si CompOs[i] = VMOs[i] alors ils correspondront automatiquement à la même localisation réelle puisque l'indice qu'ils stockent est identique.

Transformer CompLoc, CompOs, VMLoc et VMOs peut être automatisé sans trop de difficultés.

Pour les OS, obtenir une liste de tous les OS les plus connus ne semble pas impossible.

Du côté des localisations, il faudrait bien préciser quelles localisations sont autorisées. Par exemple, est-ce qu'on parle d'un état des Etats-Unis ou des Etat-Unis, est-ce qu'on peut indiquer un département ou est-ce qu'on reste à l'échelle des pays. Cela limitera peut-être le client dans ses demandes mais cela semble être une contrainte obligatoire à placer.

Malheureusement, le fait que nous n'arrivons pas à implémenter la contrainte qui fix les valeurs de Y (la variable qui nous indique combien de composants sont sur une machine) nous empêche de tester toutes les contraintes.

Nous avons donc essayé en nous contentant des contraintes sur les composants. En effet, Y n'est utile que pour la fonction de coût qu'on cherche à minimiser.

Mais lorsqu'on essaie, il ne se passe rien, Minizinc ne renvoie pas d'erreurs et ne nous affiche aucune solution. Nous n'avons pas réussi à comprendre cette erreur.

7/Organisation

Dans un premier temps nous avons organisé des réunions entre nous et nos professeurs encadrants une fois toutes les deux semaines.
Puis étant donné que nous bloquions sur l'implémentation du modèle, nous n'avons plus trop donné de nouvelles et une seule réunion a été organisée depuis la soutenance de mi-parcours, à la demande de notre professeur encadrant.

De notre côté nous avons utilisé un dossier sur google drive pour stocker tous les fichiers et pour rédiger le rapport, le diaporama ainsi que toutes nos prises de notes. De cette façon nous pouvons travailler tous ensemble sur les fichiers et avons accès aux notes de nos partenaires.

Nous avons aussi créé un serveur discord pour échanger entre nous de façon à ce que, de la même façon, chacun puisse voir tous nos messages. Au besoin on peut s'en servir pour se transmettre des fichiers et surtout comme salon de discussion vocal.

Nous avons surtout utilisé le dépôt git pour gérer les différents codes. Il y a également les papiers que nous avons lus et le diapo de la soutenance de mi-parcours.

L'implémentation s'est essentiellement passée en pair programming afin de réduire le nombre de conflits git étant donné que les fichiers sont très courts. Nous avons aussi décidé de travailler en présentiel à l'université, cela nous a permis de nous entraider et de se concerter même si tout le monde ne travaillait pas sur le TER, de pouvoir utiliser un tableau si un problème persistait et qu'on avait besoin de tous se pencher dessus.

8/ Conclusion

Ce TER nous a permis d'entrevoir le monde de la recherche, ses difficultés, son intérêt et sa manière de fonctionner. Cela nous a aussi demandé de faire face à de nombreux imprévus niveau organisation et travail. Nous nous sommes rendu compte que réaliser un projet de quelques semaines et un projet de plusieurs mois était totalement différent. Et avons dû composer avec ces imprévus afin de pouvoir avancer

9/ Bibliographie

BtrPlace: A Flexible Consolidation Manager for Highly Available Applications, 2013, Fabien Hermenier, Julia Lawall, and Gilles Muller

Virtual Machine Placement for Hybrid Cloud using Constraint Programming, 2017, Hélène Coullon, Guillaume Le Louet, Jean-Marc Menaud

A Cloud Brokerage Solution: Formal Methods Meet Security in Cloud Federations, 2018, Salwa Souaf, Pascal Berthomé, Frédéric Loulergue