Mathieu Bivert, CSSR, bivert@essi.fr

PFE : Rendu intérmédiaire D_2 (draft)

Mars 2013

Placement constraints for a better QoS in clouds

POLYTECH NICE-SOPHIA

Entreprise Université de Nice-Sophia Antipolis

Lieu Sophia-Antipolis, France

Responsable Fabien Hermenier, équipe OASIS, fabien.hermenier@unice.fr

Résumé

Ce document présente une formalisation du typage des nœuds et des machines virtuelles dans le cadre du logiciel Entropy. Une implémentation y est décrite succintement.

Abstract

This document describes a formalisation of the nodes and virtual machines typing within the framework of Entropy. A implementation is also succintly described.

Table des matières

1 Introduction

1.1 Généralités et vocabulaire

BtrPlace est un logiciel permettant de répartir efficacement un ensemble de machines virtuelles (VMs) sur un ensemble de nœuds en respectant un ensemble de contraintes donnés par l'utilisateur. Pour ce faire, BtrPlace modélise les actions sur les VMs et nœuds (comme l'éteignage, l'allumage, etc.) via des intervalles de durées appellés slices. Le but de cette partie du travail est de typer VMs et nœuds, afin de refléter les différents hyperviseurs présents sur le marché.

Plus formellement :

Type entier $t \in \mathcal{T}$ associé à chaque système de virtualisation, par exemple, KVM= 0, VMWare= 1, ...

Nœud serveur physique, noté $n \in \mathcal{N}$,doté d'un type courant $T(n)$ et d'un ensemble de types possibles $\mathcal{T}_n$;

VM machine virtuelle, notée $v \in \mathcal{V}$, à laquelle est associée un type fixe $T(v) \in \mathcal{T}$ et un emplacement courant $P(v) \in \mathcal{N}$;

Déploiement opération de redémarrage de nœud, éventuellement accompagnée d'un changement de type pour le nœud.

Reconfiguration opération durant laquelle BtrPlace change le placement des VMs sur les nœuds, en fonction des contraintes établies par l'utilisateur ;

Slices la modélisation des actions de reconfiguration [FH12] est réalisée à l'aide de *slices*, qui correspondent à une durée finie pendant un processus de reconfiguration, durant laquelle des ressources sont utilisées. On distingue deux types de slices :

consuming slice , $c \in \mathcal{C}$, où les ressources sont utilisées au début de la reconfiguration ;

demanding slice , $d \in \mathcal{D}$, où les ressources sont utilisées à la fin de la reconfiguration ;

La fonction T associe à une VM ou un nœud son type ; la fonction P associe à une VM ou une slice un nœud.

Un nœud est doté d'une nouvelle dimension de type. Celle-ci est booléenne : soit le type change, auquel cas, la valeur est de 1, sinon, elle vaut 0. Dans les graphes suivants, elle est représentée à part pour des questions de lisibilité.

1.2 Configuration d'exemple

Dans un premier temps, on cherche à obtenir une configuration minimaliste, mettant en œuvre suffisamment d'éléments pour représenter le problème :

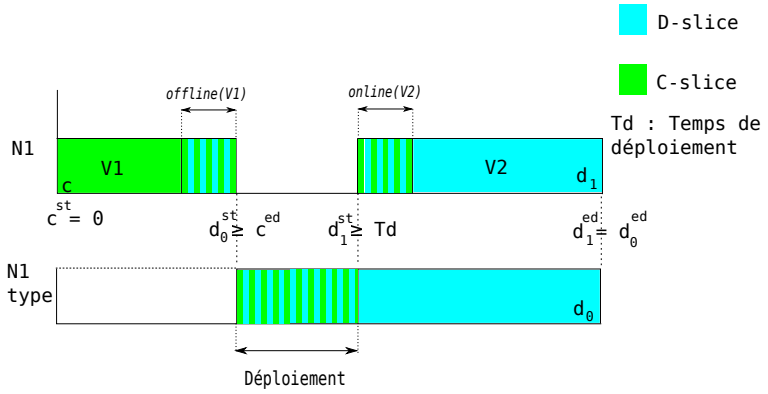

FIGURE 1 – Exemple de configuration mettant en œuvre un changement de type ; v_1 est mise hors-ligne, v_2 est allumée

Sur la figure 1, v_1 et v_2 sont deux machines virtuelles de types différents, par exemple Xen et VMWare. Pour simplifier le problème, on ne considère que des actions de démarrage et d'éteignage pour les VMs. En effet, considérer d'autres types d'actions dans cet exemple ne change pas le problème, la modélisation par des slices permettant de s'abstraire des actions précises sur VMs.

L'opération de déploiement sur le nœud n_1 se résume à :

1. mettre hors-ligne v_1 ;
2. éteindre n_1 ;
3. allumer n_1 en changeant son type, c'est-à-dire en changeant son hyperviseur.
4. démarrer v_2 ;

Le temps T_d pris par cette opération est spécifié par l'admninistrateur du datacenter dans la configuration de BtrPlace.

Dans les paragraphes qui suivent, on cherche un modèle correspondant à l'exemple du paragraphe précèdent. On commence par décrire le cas général, puis on travaille par incrément depuis un cas particulier pour arriver à une implémentation « complète ».

2 Modélisation abstraite du cas général

Pour chaque type de plateforme, un dimension est ajoutée à chaque nœud. On considère que ces dimensions sont stockés dans un vecteur $v \in \mathbb{N}^n$. La façon dont on construit v et la justification de l'ensemble de définition des dimensions ($\mathbb{N}$) sont données dans la section suivante.

Le placement est satisfait ssi au plus une seule de ces dimensions est non nulle :

$$(\exists! x \in v_i), x \neq 0 \ (0)$$

La contrainte est donc satisfaite pour un nœud n dans deux cas :

1. v_i ne contient que des 0 : aucun hyperviseur n'est actif ;
2. v_i a une composante non-nulle : un unique hyperviseur tourne sur n.

`XXX occurencemin/max deprecated?`
`https://www.iam.unibe.ch/scg/svn_repos/Students/haense/CaseStudies/choco/choco-kernel/src/main/java/`

3 Modélisation pour BtrPlace

3.1 Cas général

Pour construire les vecteurs v_i, on parcourt les nœuds utilisés dans le processus de reconfiguration pour créer table recensant les différents types d'hyperviseurs.

Dans le cas présent, la dimension $d = v_i$ peut très bien être de type booléene : si le nœud supporte le type, d est vraie, sinon d est fausse. Cependant, les licences d'utilisations des hyperviseurs spécifient parfois un nombre limités de machines virtuelles pouvant tourner. Afin de prendre en compte ce cas, il est intéressant d'utiliser une dimension entière, représentant le nombre de machines virtuelles de type t autorisées à fonctionner sur le nœud.

L'équation (0) est équivalente à l'équation suivante, exprimée en terme de contraintes :

$$OccurenceMin(v_i, 0) == len(v_i) - 1 \Leftrightarrow Occurence((v_i : len(v_i) - 1, 0, true, true)$$

OccurenceMin est deprecated mais tout de même présenté cas plus simple qu'*Occurence*. Des détails sur ce dernier sont fournis dans la section implémentation.

3.2 Cas particulier : le nouveau type d'une plateforme est connu

Lorsque le nouveau type est une propriété du modèle qui n'a pas à être déterminée par le solveur, le problème peut être simplifié. L'utilisation d'une slice pour la dimension de temps devient inutile ; on peut se contenter de deux variables indiquant les temps de début et de fin de l'opération de déploiement, respectivement notés D^{st} et D^{ed}.

Le type du nœud étant modifié, les VMs présentes au début de la reconfiguration doivent nécessairement être déplacées ou migrées, suivant les autres contraintes. Les nouvelles VMs peuvent alors être placées à l'aide de la contrainte *fence*, d'une façon similaire à ce qui se passe dans *entropy-fh/src/main/java/entropy/plan/choc*
Fence contraint un ensemble de VMs à tourner sur un ensemble de nœuds ; donc dans le cas présent, limite le placement des VMs au nœuds ayant le même type.

Pour satisfaire le placement sur un nœud n, deux contraintes supplémentaires sont données au solveur :

1. Les anciennes VMs partent avant le début de l'opération de redéploiement, c'est-à-dire,

$$(\forall c \in \mathcal{C}), P(c) = n \Rightarrow c^{\text{ed}} \leq D^{\text{st}}$$

2. Les nouvelles VMs arrivent une fois le redéploiement terminé, c'est-à-dire :

$$(\forall d \in \mathcal{D}), P(d) = n \Rightarrow d^{\text{st}} \geq D^{\text{ed}}$$

Le placement est satisfait ssi chaque VM est bien placée sur un nœud de même type, ie. :

$$(\forall v \in \mathcal{V}), (\exists n \in \mathcal{N}), P(v) = n \Rightarrow T(n) = T(v)$$

Cette contrainte doit être implémentée dans BtrPlace via Choco.

4 Implémentation

4.1 Cas particulier

```
XXX Interface: Platform({ nœud_i -> plateforme, nœud_j -> plateforme, ...})
```

On ajoute une nouvelle contrainte de placement *Platform*. Son constructeur prends en argument une *HashMap* associant les nœuds devant changer de type à leur nouvelle plateforme.

On contraint alors les VMs présentent sur le nœud à se déplacer avant le temps de début de redémarage du serveur. Pour cela, on récupère les c-slices des actions associées aux VMs, et on ajoute une contrainte sur le temps de fin de ces slices.

Enfin, si le nœud s'apprête à changer de type, on regarde toutes les d-slices entrant en jeu dans la reconfiguration. On sélectionne ensuite les d-slices dont les VMs ont le même type que le nœud à la fin du processus. Finalement, on contraint ces slices à ne démarrer qu'après la fin du processus de retypage, c'est-à-dire une fois que le nœud a bien redémarré et changé d'hyperviseur.

Le code implémentant la contrainte est disponible via le fichier *entropy-src/main/java/entropy/vjob/Platform.java*.
Cette classe est testée en suivant des use-cases définis par Fabien Hermenier dans *entropy-fh/src/test/java/entropy/vjob/co*

4.2 Cas général

La documentation d'*Occurence* :

```
public Occurrence(IntDomainVar[] vars,
                  int occval,
                  boolean onInf,
                  boolean onSup)

    Constructor, API: should be used through the Problem.createOccurrence API Define
an occurence constraint setting size{forall v in lvars | v = occval} <= or >= or = occVar
assumes the occVar variable to be the last of the variables of the constraint:
vars = [lvars | occVar] with lvars = list of variables for which the occurence of
```

```
occval in their domain is constrained

    Parameters:
        occval - checking value
        onInf - if true, constraint insures size{forall v in lvars | v = occval} <= occVar
        onSup - if true, constraint insure size{forall v in lvars | v = occval} >= occVar
```

5 Validation

Les implémentations précèdentes sont validées pas des tests unitaires mettant en œuvre des uses-cases significatifs vus avec l'encadrant.

Références

[FH12] Gilles Muller Fabien Hermenier, Julia Lawall. Btrplace : A flexible consolidation manager for highly available applications. 2012.