RenPar'20 / SympA'14 / **CFSE 8**
Saint-Malo, France, du 10 au 13 mai 2011

Auto-configuration d'applications réparties dans le nuage *

Xavier Etchevers[*], Thierry Coupaye[*], Fabienne Boyer[**], Noël de Palma[**]

[*] Orange Labs
28 Chemin du Vieux Chêne - BP 98
F-38243 Meylan
{xavier.etchevers, thierry.coupaye}@orange-ftgroup.com

[**] Université Joseph Fourier
BP 53
F-38041 Grenoble Cedex 9
{fabienne.boyer, noel.de_palma}@inrialpes.fr

Résumé
Dans le domaine de l'informatique dans le nuage, les offres actuelles de type Platform as a Service (PaaS), c'est-à-dire d'environnements supportant les différentes phases du cycle de vie d'une application, demeurent spécifiques à un domaine applicatif et ne sont que partiellement automatisées. Cette limitation résulte de l'absence d'un modèle architectural permettant de décrire une application répartie en y incluant les piles logicielles (système d'exploitation, intergiciels, application) qui composent les machines virtuelles et leurs interdépendances en termes de configuration.
Cet article présente (i) un modèle d'application à base de composants permettant de décrire toute application répartie sous la forme d'un ensemble de machines virtuelles interconnectées, (ii) une chaîne automatisée permettant de déployer une application répartie dans le nuage. Cette chaîne comprend notamment un protocole d'auto-configuration décentralisé des machines virtuelles composant une application, (iii) une première évalution de performance qui montre la viabilité de la solution.

Mots-clés : informatique dans le nuage (cloud computing), applications réparties, virtualisation, déploiement, informatique autonome

1. Introduction

Les environnements de type *informatique dans le nuage* [1] se déclinent en trois grands niveaux d'offre selon le type de ressource mis à disposition. Le niveau Infrastructure as a Service (IaaS) a pour but de permettre d'accéder à des ressources matérielles (calcul, stockage, réseau) virtualisées. La couche Software as a Service (SaaS), vise à exposer des applications logicielles à destination des utilisateurs finaux. La couche intermédiaire, Platform as a Service (PaaS), offre un ensemble d'outils et d'environnements d'exécution qui permettent de gérer le cycle de vie des applications. Ce cycle de vie comprend notamment les phases de conception, développement, déploiement et plus généralement d'administration des applications (gestion de la charge, de la tolérance aux pannes, de la sécurité, etc.).
Cet article traite du déploiement d'applications réparties dans des environnements virtualisés de type *informatique dans le nuage*. Pour de tels déploiements, il est nécessaire de générer les images virtuelles qui seront instanciées sous forme de machines virtuelles qui supporteront l'exécution de l'application sur une plate-forme de type IaaS. Chaque image se compose de briques techniques (système d'exploitation, intergiciels), et de briques fonctionnelles (données et logiciels applicatifs). Une fois instanciée, chaque machine virtuelle fait généralement l'objet d'une phase de configuration dynamique finalisant le paramétrage de l'application répartie.
Chaque machine virtuelle comprend en effet une myriade de paramètres de configuration relatifs à l'ensemble de sa pile logicielle. Certains concernent des aspects de configuration locale (e.g. taille d'un

*. Ce travail a bénéficié d'une aide de l'Agence Nationale de la Recherche portant la référence ANR-08-SEGI-017

pool, données d'authentification), d'autre participent à la définition d'interconnexions entre éléments distants (e.g. adresse IP et port d'accès à un serveur). Selon leur nature, ces paramètres peuvent être renseignés pendant la génération de l'image ou après l'instanciation des machines virtuelles. L'une des complexités majeures du déploiement d'une application distribuée dans le nuage réside dans le nombre de paramètres à renseigner, dans leur variété et dans le moment où ils peuvent / doivent être renseignés.
De manière générale, les solutions de déploiement actuellement disponibles ne prennent pas en charge ces différents paramètres de configuration, qui restent majoritairement gérés au travers de scripts dédiés. De plus, ces solutions ne sont pas capables de traiter, de façon automatisée, la génération des images, leur instanciation sous forme de machines virtuelles et la configuration de celles-ci indépendamment du type d'application répartie déployée. Par exemple, la solution Google App Engine [2] ne considère que les services Web organisés en tiers bien identifiés. L'absence de solution générale provient essentiellement, selon nous, d'un manque de formalisme permettant de décrire l'architecture d'une application répartie et ses contraintes de configuration dans une infrastructure virtualisée de type informatique dans le nuage.
Cet article propose une solution générale, appelée VAMP pour Virtual Applications Management Platform, automatisant le déploiement d'applications réparties arbitraires au sein du nuage. L'approche proposée est architecturale en ce sens qu'elle se base sur une représentation explicite de l'architecture répartie des applications. Nous proposons d'une part un formalisme permettant de décrire une application comme un ensemble de machines virtuelles interconnectées, et d'autre part un moteur permettant d'interpréter ce formalisme et d'automatiser le déploiement effectif de l'application sur une plate-forme de type IaaS.
Les contributions de cet article sont plus précisément :
– un formalisme permettant d'avoir une vision globale de l'application à déployer en termes de composants, de contraintes de configuration et d'interconnexions ainsi que de leur répartition en machines virtuelles. Ce formalisme étend le langage OVF qui permet de décrire des machines virtuelles avec un langage de description d'architecture [3] (ADL en anglais) qui permet de décrire l'architecture logicielle d'une application répartie,
– un moteur de déploiement, c'est-à-dire un support d'exécution capable de déployer une application décrite dans ce formalisme de manière automatisée. Ce moteur se base sur un protocole décentralisé d'auto-configuration des différentes machine virtuelles instanciées - à même selon nous de faciliter le passage à l'échelle de la phase de configuration dynamique,
– une évaluation de performances de la solution proposée sur une plate-forme IaaS industrielle (Eucalyptus [4]).
Cet article est organisé de la façon suivante. La section 2 décrit le formalisme que nous proposons pour modéliser une application à déployer dans le *nuage*. La section 3 présente la chaîne d'automatisation du déploiement de l'application et la section 4 se focalise sur la description du protocole d'auto-configuration décentralisé. Une évaluation de ce support suit en section 5. La section 6 propose un positionnement des travaux traitant du déploiement d'applications dans le nuage. La section 7 conclut en présentant les perspectives de ces travaux.

2. Modèle d'application virtualisée

Cette section introduit le modèle OVF étendu proposé pour la description des applications.

2.1. Principes d'extension d'OVF

OVF (Open Virtualization Format) est un standard en cours de définition proposé par la Distributed Management Task Force (DMTF) [5]. Il s'agit d'un formalisme déclaratif, extensible, basé sur XML, qui permet de décrire l'organisation du déploiement de plusieurs images virtuelles en caractérisant explicitement les machines virtuelles dans lesquelles elles doivent être instanciées. Il est ainsi possible de décrire les capacités de calcul, de stockage et de réseau de chaque machine virtuelle, leur ordre de démarrage, les besoins en matière de redémarrage (reboot) après reconfiguration. OVF permet également de définir des variables dont la valeur sera renseignée statiquement ou dynamiquement (au moment du déploiement) par l'utilisateur ou par l'environnement d'exécution.
Le formalisme proposé apporte deux extensions à OVF :

– La première consiste à définir une nouvelle section (*AppArchitectureSection*) dans le format. Son but est d'offrir une vue architecturale de l'application distribuée contenue dans le package OVF. Le formalisme choisi pour décrire l'architecture applicative au sein du descripteur OVF est un ADL.
– La seconde est une modification de la section *References* consacrée aux ressources référencées par le package OVF. Cette section inclut notamment la définition des images nécessaires à l'instanciation des machines virtuelles. Il s'agit cependant de ressources statiques au sens où elles doivent exister au moment de la création du package OVF. Or, afin de pouvoir automatiser l'ensemble de la phase de déploiement d'une application dans le nuage, et plus précisément la création des images logicielles, il est nécessaire de pouvoir décrire une ressource non encore disponible pour la générer à la volée. L'ajout de l'élément *DynamicImage* permet de définir les éléments qui participent à la génération de l'image, c'est-à-dire le système d'exploitation, les intergiciels et les composants applicatifs ainsi que les données qu'elle doit embarquer.

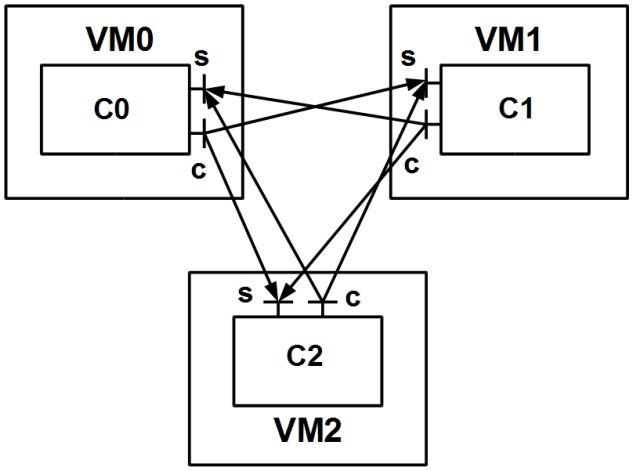

FIGURE 1 – Exemple élémentaire d'application répartie

Dans la mesure où le formalisme proposée s'appuie sur OVF, il possède la même propriété d'extensibilité et offre ainsi un pouvoir d'expression équivalent.

2.2. Description des applications réparties

La description des applications réparties, telle qu'attendue dans la section (*AppArchitectureSection*) du format OVF étendu, repose sur le langage de description d'architecture du modèle à composants Fractal [6] [7]. Ce langage permet de préciser la structure globale d'une application en termes de composants, de configuration et d'interconnexions [2]. Un composant comprend un ensemble d'interfaces qu'il peut soit exposer, on parle alors d'interface serveur, soit requérir, on parle alors d'interface cliente. L'interface cliente d'un composant et l'interface serveur d'un autre composant peuvent être associées au travers d'un lien. Chaque composant contient également une information de localisation (*virtual node*), qui dans le cas présent, désigne la machine virtuelle sur laquelle le composant doit être embarqué.

Ce langage fournit un niveau d'abstraction supérieur à des scripts de configuration spécifiques, ce qui permet d'automatiser les opérations de configuration de l'ensemble de la pile logicielle embarquée dans une machine virtuelle. D'autre part, il permet de prendre en compte la modélisation d'applications patrimoniales en considérant les composants comme des adaptateurs de configuration [8].

La figure 2 fournit un exemple de descripteur OVF étendue pour l'application élémentaire *TokenApp* décrite dans la figure 1. Cette application est constituée de trois composants interconnectés, nommés respectivement *C0*, *C1* et *C2*. Chacun d'eux possède une interface serveur *s* et une collection d'interfaces

2. Le modèle Fractal est également récursif (un composant peut être composé de sous-composants de façon hiérarchique à un niveau arbitraire) et réflexif (l'architecture est explicite et manipulable lors de l'exécution)

```
<Envelope ...>
  <References>
    <!-- Fichier statique contenant le jar de VAMP -->
    <File ovf:id="vampJar"  ovf:href="vamp.jar"/>
    <!-- Définition d'une image dynamique -->
    <DynamicImage ovf:id="appDisk" pm:version="1.0" ovf:required="false">
      <OperatingSystem pm:id="Debian" pm:version="5.0" pm:architecture="i386"/>
      <Products>
        <Product pm:id="Java" pm:version="1.6"/>
        ...
      <Products>
    </DynamicImage>
  </References>
  <DiskSection>
    <Disk ovf:diskId="appDiskId" ovf:capacity="1024"
          ovf:capacityAllocationUnits="byte * 2^20"
          ovf:format="http://xen" ovf:fileRef="appDisk"/>
  </DiskSection>
  ...
  <!-- Architecture applicative -->
  <AppArchitectureSection>
    <definition name="TokenApp">
      <component name="C0">
        <interface name="c" role="client" .../>
        <interface name="s" role="server" .../>
        ...
        <virtual-node name="VM0"/>
      </component>
      <component name="C1">
        ...
      </component>
      <component name="C2">
        ...
      </component>
      <binding client="C0.c" server="C1.s" />
      <binding client="C0.c" server="C2.s" />
      ...
    </definition>
  </AppArchitectureSection>
  ...
  <!-- Configuration des machines virtuelles -->
  <VirtualSystemCollection ovf:id="App">
    <!-- Configuration de VM0 -->
    <VirtualSystem ovf:id="VM0" rsrvr:min="1" rsrvr:max="1">
      ...
      <VirtualHardwareSection>
        <Item>
          <rasd:ElementName>Harddisk 1</rasd:ElementName>
          <rasd:HostResource>ovf://disk/appDiskId</rasd:HostResource>
          ...
        </Item>
        ...
      </VirtualHardwareSection>
    </VirtualSystem>
    ...
  </VirtualSystemCollection>
</Envelope>
```

FIGURE 2 – Exemple de descripteur OVF étendu, incluant l'architecture globale de l'application dans la section *AppArchitectureSection*. Les éléments en gras sont les extensions proposées par VAMP

cliente *c*. Chaque composant est déployé sur une machine virtuelle distincte. Chaque composant est interconnecté aux deux autres via des liens entre les interfaces clientes et serveurs.

3. Chaîne automatisée de déploiement

Sur la base du formalisme présenté dans la section précédente, cette section propose une chaîne de traitement permettant d'automatiser le déploiement d'une application répartie dans le nuage.

Comme l'illustre la figure 3, elle procède selon la séquence d'étapes suivante :

Génération des images : la première étape consiste à créer, à partir d'une description d'application selon le formalisme décrit en section 2, l'ensemble des images correspondant aux machines virtuelles constituant l'application [3].

Publication des images : dans un deuxième temps, les images ainsi générées sont publiées au sein d'une plate-forme d'IaaS. Il s'agit de les enregistrer dans le système de stockage de la plate-forme et de les rendre accessibles au travers de son répertoire d'images.

Instanciation des images : lors de cette étape les images publiées sont instanciées dans la plate-forme d'IaaS, sous forme de machines virtuelles. Dès lors, chaque machine virtuelle démarre indépendamment et en fin de boot instancie automatiquement le configurateur qu'elle embarque. Les étapes d'auto-configuration peuvent alors commencer.

Configuration locale et globale : chaque configurateur procède à la configuration dynamique de l'application, c'est-à-dire de l'ensemble des paramètres locaux relatifs aux éléments logiciels qui constituent la machine virtuelle ainsi qu'à la configuration des dépendances avec d'autres machines virtuelles.

Activation de l'application : Enfin, chaque configurateur participe au démarrage de l'application. Il assure l'activation des composants applicatifs embarqués au sein de la machine virtuelle dont il a la charge tout en tenant compte des contraintes de démarrage qui leurs sont associées.

D'un point de vue conceptuel, l'étape de génération automatisée d'images est réalisée par une entité appelée le gestionnaire d'images. VAMP implémente dès à présent ce gestionnaire d'images. Ceci consiste principalement à traduire les informations contenues dans la section *DynamicImage* du descripteur OVF étendu en accès programmatiques au service UForge d'Usharesoft [9]. Outre les éléments logiciels nécessaires au fonctionnement de l'application (système d'exploitation, intergiciels, binaires de l'application), VAMP ajoute dans chaque image tout ou partie des propriétés de configuration indépendantes de l'environnement d'exécution. Enfin, pour disposer de machines virtuelles auto-configurables, VAMP y adjoint également une représentation locale de l'architecture applicative, c'est-à-dire réduite aux composants et des liens dont elle a la responsabilité, ainsi qu'un configurateur chargé d'interpréter cette représentation pour instancier, configurer et activer les composants applicatifs dont il a la charge [4]. Les étapes de publication et d'instanciation des images sont regroupées dans le gestionnaire de placement qui interagit avec la plate-forme d'IaaS au travers des APIs qu'elle expose. Ce gestionnaire confère donc à VAMP une propriété d'indépendance vis-à-vis de la couche d'IaaS. Un gestionnaire englobant appelé gestionnaire de déploiement assure l'orchestration entre ces trois premières étapes de la chaîne automatisée de déploiement.

VAMP permet de renseigner des propriétés de configuration à différents stades du processus de déploiement. Soit il s'agit de paramètres statiques dont la valeur ne dépend pas de l'environnement d'exécution. Celle-ci peut alors être renseignée par le gestionnaire de déploiement, lors de la constitution des images. Soit il s'agit de paramètres dynamiques dont la valeur n'est connue qu'après l'instanciation des machines virtuelles. Dans ce cas, celle-ci sera renseignée par un configurateur au moment de l'auto-configuration. Ainsi, par exemple, si l'IaaS dispose d'un mécanisme de pré-réservation d'adresse IP indépendant de celui assurant la création d'une machine virtuelle, il sera possible de renseigner l'adresse d'une machine virtuelle lors de la création de l'image associée. Dans le cas contraire, l'adresse IP d'une machine virtuelle ne sera connue qu'au moment de son instanciation et VAMP la propagera dans la configuration globale durant la phase de configuration dynamique. Cette phase est détaillée dans la section suivante.

4. Protocole d'auto-configuration et de démarrage

La seconde contribution de cet article est un mécanisme asynchrone et décentralisé d'auto-configuration et d'activation des composants applicatifs. L'auto-configuration est pilotée par les configurateurs au sein de chaque machine virtuelle. Chacun d'eux assure trois fonctions :

3. Les images générées présentent une empreinte minimale, c'est-à-dire qu'elles n'embarquent que les paquetages indispensables à leur fonctionnement.

4. Durant la phase d'auto-configuration, le configurateur utilise la définition locale de l'architecture applicative pour connaitre les composants qu'il doit instancier et leurs besoins en termes de configuration locale et globale.

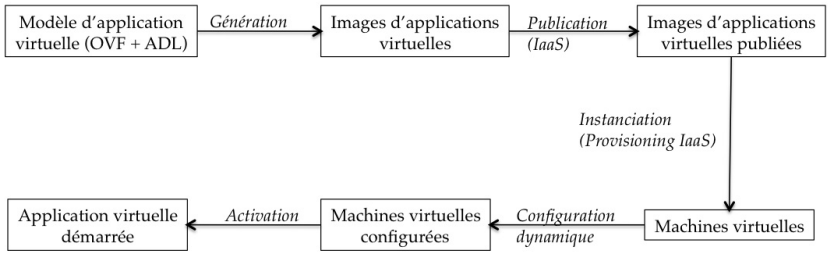

FIGURE 3 – Processus automatisé de déploiement d'une application dans le nuage avec VAMP

1. à partir de la définition architecturale locale dont il dispose, il instancie les composants applicatifs dont il a la responsabilité et les configure localement. Ceci consiste principalement à inférer les liens entre les composants déployés au sein de la machine virtuelle.
2. il participe à la configuration globale de l'application en inférant les liens distants. Il s'agit des liens entre l'interface d'un composant colocalisé dans la même machine virtuelle que le configurateur considéré et l'interface d'un composant non colocalisé avec ce même configurateur (un composant déployé au sein d'une autre machine virtuelle et donc géré par un autre configurateur).
3. il assure le démarrage des composants applicatifs dont il a la charge (ceux qui sont colocalisés avec lui au sein d'une même machine virtuelle) selon un ordre qui est fonction des dépendances entre composants applicatifs. Dans VAMP, ces dépendances sont restituées au travers d'une propriété appelée *contingence*, relative à une interface cliente d'un composant C. Elle permet d'indiquer si l'interface cliente considérée doit être liée à une interface serveur, au travers d'un lien, pour que le composant C puisse être activé. On parle alors de contingence obligatoire. Dans le cas contraire, la contingence est dite optionnelle. Par extension, la contingence d'un lien désignera la contingence de son extrémité cliente. Ainsi, un composant possède un lien obligatoire ou optionnel si l'extrémité cliente du lien considéré désigne l'une des interfaces du composant.

Le caractère décentralisé vise à éliminer de potentielles ressources mutualisées synonymes de risques en termes de passage à l'échelle, tant d'un point de vue des performances (goulot d'étrangement) que de la fiabilité (criticité d'une ressource). Le côté asynchrone, quant à lui, offre une plus grande agilité, chaque composant pouvant démarrer indépendamment et contribuer ainsi au fonctionnement de l'application en mode dégradé.

Les deux dernières étapes nécessitent l'établissement de communications entre configurateurs. Ces échanges prennent la forme de messages envoyés au travers d'un bus asynchrone et fiable, réparti sur l'ensemble des machines virtuelles. Le caractère asynchrone en mode déconnecté permet d'envoyer des messages à un destinataire qui n'existe pas encore ou qui a disparu temporairement suite à une panne. La propriété de fiabilité assure la persistance des messages émis par un émetteur jusqu'à leur traitement par les destinataires, même si l'émetteur ou les destinataires tombent en panne entre temps. VAMP s'appuie sur l'implémentation AAA de bus à messages asynchone, fiable et distribué [10].

Les phases d'élaboration des liens applicatifs distants et de démarrage s'organisent selon le protocole suivant :

1. Pour chaque lien λ associé à un composant dont il a la charge et dont l'extrémité serveur est locale,[5] chaque configurateur γ envoie au configurateur γ' en charge de l'extrémité cliente de λ un message qui contient la référence de l'extrémité serveur (*bind*). Cette référence contient l'ensemble des informations requises par le composant client pour interagir avec le composant serveur.
2. Lorsqu'un configurateur reçoit un message contenant une telle référence, ceci signifie qu'il a la charge de l'extrémité cliente du lien. Il procède alors à la configuration locale du lien.

5. Par simplification de langage, l'extrémité d'un lien est dite locale (respectivement distante) s'il s'agit d'une interface d'un composant local (respectivement distant.

3. Une fois l'ensemble des références serveurs émises, un configurateur peut lancer le processus de démarrage des composants applicatifs. Cela consiste à activer, dans un premier temps, tous les composants applicatifs ne possédant pas de liens dont la contingence est obligatoire.
4. Le configurateur détermine alors, pour chaque composant activé, la liste des liens dont l'extrémité serveur est locale. Pour chacun d'eux, il envoie à chaque configurateur, en charge de l'extrémité cliente associée, un message de démarrage (*start*).
5. A la réception d'un message de démarrage, un configurateur détermine si le composant destinataire du message dispose alors de l'ensemble des références serveurs de ses liens obligatoires. Si tel est le cas, le composant est activé le configurateur exécute l'étape 4 sur ce composant. De proche en proche, l'ensemble de l'application est démarrée.

La figure 4 illustre un exemple d'exécution du protocole d'auto-configuration et d'activation de VAMP dans le cas de l'application précédemment présentée.

5. Evaluation

L'objectif de cette évaluation est d'estimer la viabilité du processus de déploiement d'une application à l'aide VAMP. Elle se focalise uniquement sur la vitesse de déploiement.
L'application TokenApp déployée au cours de ces tests est constituée de N composants applicatifs identiques (voir figure 1). Toutes les interfaces clientes de chaque composant présentent une contingence optionnelle. La structure totalement maillée entre composants constitue l'architecture applicative la plus défavorable vis-à-vis d'un processus d'auto-configuration. Le contenu des messages d'élaboration de liens se compose d'une dizaine d'octets stockés sur le disque, au moment de leur réception côté client. Aucun des composants applicatifs n'est colocalisé avec un autre composant. Ainsi chacun d'eux est déployé dans une machine virtuelle qui lui est propre et qui comporte un cpu virtuel et une mémoire virtuelle de 128 Mo. La taille de l'image associée à chaque machine virtuelle est de 1 Go.

5.1. Environnement de test

L'environnement de validation utilisé est une plate-forme privée d'informatique dans le nuage. Elle est totalement isolée tant d'un point de vue réseau que vis-à-vis de son utilisation. Cette isolation garantie que les mesures de performances obtenues n'ont pas fait l'objet d'effets de bord résultant d'un partage des ressources physiques avec d'autres applications. Cette plate-forme est constituée de 6 machines de type Dell Studio Hybrid (1 x Intel Core 2 Duo T8100 2.1 GHz, RAM 4 Go, HDD 320 Go) reliées

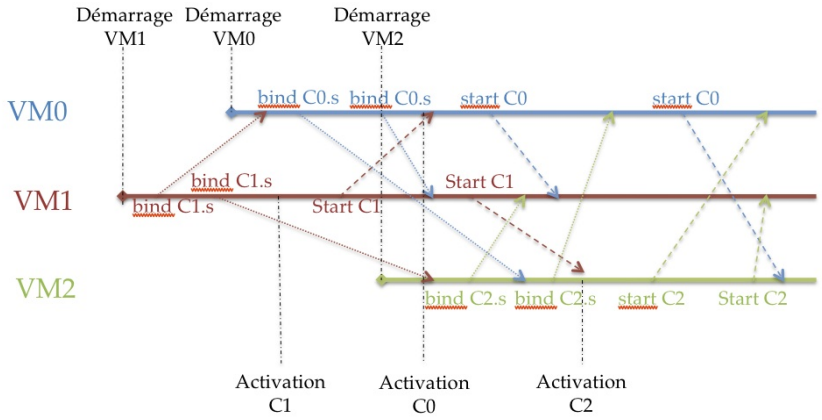

FIGURE 4 – Exemple de configuration et d'activation automatisées avec VAMP : le configurateur embarqué dans VM1 envoie les références de ses interfaces serveurs aux configurateurs avec lesquels il partage des liens. Il n'a cependant pas besoin d'attendre que ceux-ci soient instanciés pour leur adresser des messages. Une fois démarré, il notifie les configurateurs qui présentent des dépendances, en termes de contingence, vis-à-vis de lui.

par un réseau local Ethernet 100 Mb/s. Sur chaque machine sont installés un système d'exploitation Linux (Debian Lenny), un hyperviseur (Xen v3.2) et la plate-forme d'IaaS partiellement open source Eucalyptus (v1.6.2). L'une des machines physiques sert de frontal Eucalyptus. Elle regroupe l'ensemble des entités d'administration de l'IaaS à savoir le *cloud controller*, le *cluster controller*, le *storage controller* et le système de stockage *Walrus*. Les autres serveurs physiques exécutent un *node controller* et sont autant de nœuds d'instanciation (ou nœuds de déploiement) de machines virtuelles. Dans le contexte de ces expérimentations, le répertoire d'images de la plate-forme d'IaaS est centralisé. Néanmoins, lors de l'instanciation d'une machine virtuelle, une copie de l'image associée est transférée puis stockée au sein du nœud de déploiement cible. En effet, la plate-forme d'IaaS ne propose pas de mécanisme de stockage en réseau NAS (Network Attached Storage en anglais) pour conserver les copies d'images instanciées.

Eucalyptus implémente les APIs EC2 [11] et S3 [12] d'Amazon Web Service (AWS). Parmi les opérations qu'elles proposent, voici celles utilisées au cours de l'évaluation :

- convertir une image stockée au format Xen sur un système de fichiers standard dans un format exploitable par l'IaaS ;
- transférer le résultat de la conversion dans le système de stockage de l'IaaS ;
- référencer un élément ainsi stocké dans le registre des images ;
- instancier sur un nœud de déploiement une nouvelle machine virtuelle à partir d'une image du registre ;
- détruire une instance de machine virtuelle ;
- déréférencer une image du registre d'images ;
- détruire un élément de la zone de stockage de l'IaaS.

5.2. Résultats

Les métriques mesurées au cours de l'évaluation sont un ensemble de durées représenté dans la figure 5. Elles ont été mesurées pour chaque machine virtuelle instanciée. L'évaluation menée s'est déroulée en plusieurs étapes. La première consiste, pour chaque métrique, à la quantifier et à valider les prévisions quant à la tendance de son évolution. La seconde mesure la corrélation ente la quantité de mémoire

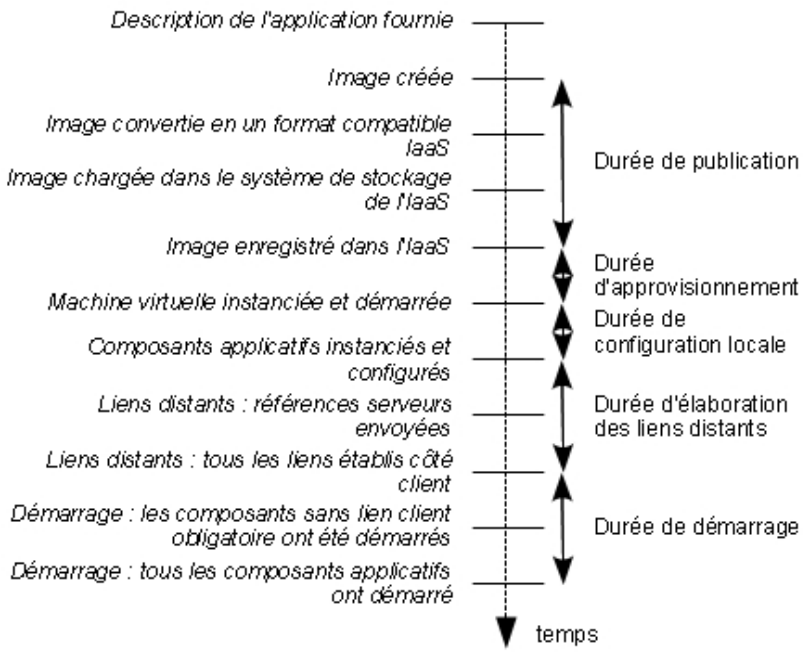

FIGURE 5 – Métriques évaluées pour estimer la viabilité du mécanisme de déploiement proposé par VAMP

disponible au sein d'une machine virtuelle et les durées mesurées. Enfin, la troisième étudie la durée de déploiement d'une application ressentie par l'utilisateur.

5.2.1. Quantification et estimation de la tendance

Les mesures ont été effectuées en faisant varier le nombre de composants applicatifs déployés (c'est-à-dire le nombre de machines virtuelles instanciées) et le nombre de nœuds d'instanciation (2 puis 5).
La valeur moyenne de la durée de publication d'une image dans l'IaaS varie de façon importante en fonction du nombre de publications simultanées. De 115 s. par image pour une publication isolée, elle atteint 200 s. par image pour 20 publications simultanées. Cette tendance plaide en faveur d'une factorisation des images publiées associée à des opérations d'auto-configuration post-instanciation plus nombreuses. Par la suite, la durée d'approvisionnement a été mesurée dans le cadre du déploiement de N machines virtuelles, chacune d'elles se basant sur une image distincte publiée préalablement dans l'IaaS. Au-delà d'une vingtaine d'instanciations en parallèle, le temps d'approvisionnement augmente de façon exponentielle du fait de la saturation du frontal d'IaaS. Afin de maintenir des conditions de test acceptables, l'hypothèse a été faite que l'application se limitait à une seule image instanciée plusieurs fois (en fonction de la valeur de N). Cette hypothèse implique qu'aucune opération de configuration statique spécifique à une machine virtuelle ne soit réalisée au cours de la génération de l'image associée. La figure 6 illustre les résultats obtenus pour la durée d'approvisionnement [6] . Cette durée d'approvisionnement dépend d'une part du nombre de machines virtuelles déployées sur chaque nœud d'instanciation et d'autre part du temps de transfert des images entre la zone de stockage de l'IaaS et les nœuds d'instanciation. Ces deux éléments impactent respectivement la contention des nœuds d'instanciation et la contention du réseau. Leurs effets sur les résultats obtenus sont antagonistes. Ainsi, le fait que la courbe relative à 2 nœuds d'instanciation ait une pente plus forte que la courbe relative à 5 nœuds d'instanciation, démontre la prépondérance de la contention des nœuds d'instanciation vis-à-vis de la contention du réseau. Cependant, pour un nombre réduit de machines virtuelles, la durée d'approvisionnement sur 2 nœuds d'instanciation est plus faible que sur 5. En effet, dans ce cas, les nœuds d'instanciation ne sont pas fortement sollicités et la contention du réseau devient le facteur limitant : le temps de transfert des images sur le réseau augmente avec le nombre de nœuds d'instanciation.

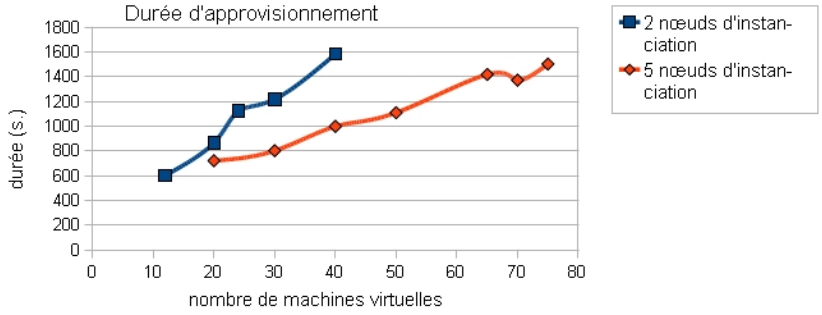

FIGURE 6 – Durée d'approvisionnement moyen d'une machine virtuelle en fonction du nombre de machines virtuelles déployées sur deux (P2) puis cinq (P5) nœuds d'instanciation

La figure 7 illustre quant à elle les résultats concernant la durée de configuration locale et la durée d'élaboration des liens distants. La première est pratiquement constante vis-à-vis du nombre total de machines virtuelles alors que la seconde en dépend linéairement, étant corrélée au nombre de liens établis par un configurateur. A l'inverse, la répartition des machines virtuelles sur les différents nœuds d'instanciation a un impact plus important sur la durée de configuration locale que sur la durée d'élaboration des liens distants.

6. Cette durée, qui est relative à la plate-forme d'IaaS sous-jacente, inclut le temps de transfert de l'image depuis la zone de stockage de l'IaaS jusqu'au nœud d'instanciation ainsi que le temps de boot de la machine virtuelle

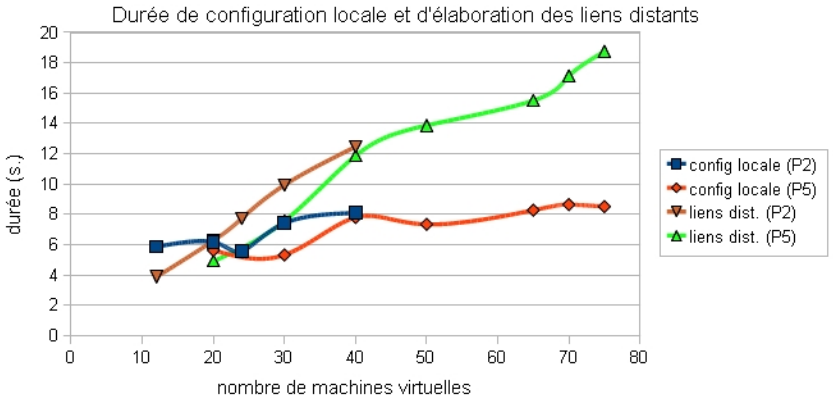

FIGURE 7 – Durée moyenne de configuration locale et d'élaboration des liens distants d'un configurateur en fonction du nombre de machines virtuelles déployées sur deux (P2) puis cinq (P5) nœuds d'instanciation

Concernant la durée de démarrage, sa valeur est tellement négligeable par rapport à l'ensemble des autres métriques (elle n'excède jamais 0,2 s.) que sa tendance n'a pas été étudiée.

5.2.2. Impact de la mémoire virtuelle

La seconde étape de l'évaluation se focalise sur la relation entre la quantité de mémoire virtuelle disponible et les performances. Ainsi, les machines virtuelles déployées comprennent 512 Mo de RAM, soit quatre fois plus que les machines initiales. En déployant un nombre de machines virtuelles suffisamment faible par nœud d'instanciation, afin de permettre à l'hyperviseur de toutes les conserver en mémoire sans recourir à du swap disque (cas le plus favorable pour les machines virtuelles avec 512 Mo de mémoire virtuelle), les durées mesurées demeurent pratiquement identiques (moins de 5% de différence).

5.2.3. Durée de déploiement ressentie par l'utilisateur

Cette dernière phase de tests fournit une première évaluation de la durée de déploiement globale d'une application. Elle consiste à mesurer le temps d'expérimentation qui correspond au maximum, sur l'ensemble des configurateurs, de la somme des métriques présentées sur la figure 5. Comme l'illustre la figure 8, ce temps évolue de façon semblable à sa composante principale qu'est la durée d'approvisionnement. Le temps d'exécution du protocole d'auto-configuration demeure une quantité négligeable (environ 2%) comparativement à la durée d'approvisionnement.

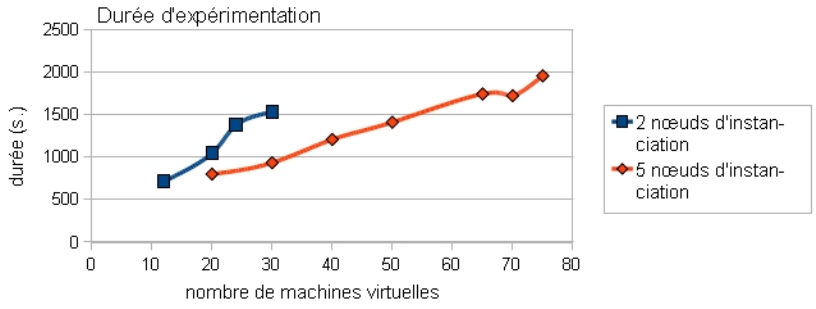

FIGURE 8 – Evaluation de la durée de déploiement d'une application perçue par l'utilisateur

6. Positionnement par rapport aux travaux existants

Les formalismes et mécanismes offerts par les solutions industrielles pour le déploiement d'applications dans le nuage sont de manière générale basiques, propriétaires, non exhaustifs et non extensibles : ils ne permettent pas de décrire finement une application répartie, ni de piloter son processus de déploiement, en particulier sa configuration statique et dynamique. En outre ces solutions présentent souvent d'importantes restrictions concernant :

– les modèles de programmation comme dans le cas de Google App Engine qui ne permet de déployer que des applications web dont le code doit se conformer à des API très spécifiques (pas de threads Java par exemple) ;
– les technologies sous-jacentes comme dans le cas de Microsoft Azure [13] qui se cantonne aux applications à base de technologie Microsoft ;
– les domaines métiers adressés comme dans le cas de Salesforce.com [14] qui se focalise sur la gestion de la relation client.

En tant qu'unique standard permettant décrire l'organisation du déploiement d'un ensemble d'images virtuelles, OVF constitue un premier pas vers la formalisation complète et cohérente d'une application distribuée déployée dans le nuage. Des acteurs majeurs tels que VMWare ou Citrix proposent d'ailleurs des plates-formes capables de déployer des paquetages au format OVF. De notre point de vue, il manque à OVF un support pour la description d'architectures réparties et pour leur configuration, en particulier la configuration dynamique des liaisons réparties. En conséquence, en l'absence de formalisme déclaratif, la configuration est enfouie dans le code lui-même des applications ou doit être réalisée par des scripts de configuration ad-hoc et externes.

[15] discute tout d'abord des implications sur la définition de l'architecture des applications réparties qui visent à être déployées dans le nuage. Il souligne particulièrement qu'une telle architecture doit être réifiée à l'exécution, ce qui constitue un point que nous partageons. Dans un second temps, il propose des éléments de langage pour la description d'architectures logicielles, de besoins vis-à-vis des plates-formes d'exécution sous-jacentes, de contraintes sur l'architecture (e.g. sur le placement et la colocalisation) et de règles d'élasticité des applications. Bien que notre travail inclue dans le futur l'expression de contraintes et le support de l'élasticité, le formalisme présenté dans le présent article ne comprend pas encore ces aspects. Sur l'expression des besoins vis-à-vis des plates-formes (IaaS) sous-jacentes, les deux approches se basent sur OVF. Sur la description d'architecture, [15] propose une approche dirigée par les modèles et propose des extensions de la syntaxe abstraite *Essential Meta-Object Facility* (*EMOF*) définie par l'initiative *Model Driven Architecture* (*MDA*) de l'*Object Management Group* (*OMG*), alors que le présent article propose d'étendre un ADL. Quant au mécanisme de déploiement (protocole et architecture), en particulier la configuration des liaisons réparties et l'ordre d'instanciation des composants, qui constitue le coeur du présent article, il n'est pas détaillé dans [15].

[16] propose une extension de *SmartFrog* [17] qui permet une allocation automatisée et optimale de ressources du nuage sur la base d'une description déclarative à la fois des composants formant une application répartie et des ressources disponibles. Les descriptions des architectures d'applications et des ressources disponibles sont exprimées dans le langage *DADL* qui permet d'exprimer des contraintes des applications sur les ressources en termes de *Services Level Agreements* (*SLAs*) et des contraintes d'élasticité. Par comparaison au présent article, [16] se focalise sur les aspects langage. DADL est une extension de SmartFrog, un framework Java pour le déploiement de systèmes répartis. L'extension de SmartFrog est réalisée par héritage de classes Java. Le langage que nous proposons est plus déclaratif et centré sur l'architecture. Il se fonde sur un formalisme bien connu de description de machines virtuelles (OVF) et intègre un langage de description d'architecture (Fractal ADL). Par ailleurs, [16] ne fournit pas de détail sur le processus de déploiement lui-même, sur ses performances ou sa robustesse. Finalement, [16] vise plus l'allocation optimale de ressources, alors que le travail abordé dans le présent article est plus axé sur l'efficacité et la fiabilité du processus de déploiement.

7. Conclusion et Perspectives

Cet article propose une solution pour le déploiement d'applications réparties dans le nuage. Une première contribution de l'article est un formalisme de description d'applications réparties sur un ensemble de machines virtuelles qui étend un formalisme (OVF) de description de machines virtuelles par un lan-

gage de description d'architecture (ADL) permettant de spécifier explicitement et de manière déclarative les composants constituant une application et les liaisons entre ces composants. Une seconde contribution est un protocole d'auto-configuration dynamique décentralisée et un moteur de déploiement supportant ce protocole et réalisant le processus complet de déploiement (en utilisant pour certaines étapes des outils, non décrit dans l'article, pour la création d'images de machines virtuelles ou l'instanciation de celles-ci). Ce protocole d'auto-configuration décentralisé est le coeur de l'article, à même selon nous, d'apporter efficacité (passage à l'échelle) et fiabilité au processus de déploiement. Une troisième contribution est une évaluation de performances qui montre la viabilité de la solution proposée sur une plate-forme IaaS industrielle.
Les propriétés du mécanisme proposé (décentralisation et asynchronisme des communications) ouvrent des perspectives intéressantes en termes de fiabilité du processus de déploiement. Au-delà de la fiabilisation du protocole de déploiement, dont une implémentation pourrait s'appuyer sur un mécanisme de reprise après panne fondé sur un stockage NAS des images instanciées, les extensions futures du travail décrit dans cet article incluent le déploiement d'applications réelles et comportant des aspects de configuration plus complexes (e.g. applications n-tiers), la fiabilité des applications déployées, i.e. l'autoréparation et plus globalement d'autres propriétés autonomiques telles que l'auto-optimisation, le support de l'élasticité des applications à la fois dans le formalisme de description et dans le support d'exécution des applications. Enfin, dans un autre registre, un support multi-IaaS constituerait une extension pertinente d'un point de vue industriel.

Bibliographie

1. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, et M. Zaharia, "Above the clouds : A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, Feb 2009. http ://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html
2. Google app engine website. http ://code.google.com/appengine/
3. N. Medvidovic et R. N. Taylor, "A framework for classifying and comparing architecture description languages," *IEEE Transactions on Software Engineering*, vol. 26, pp. 70–93, January 2000.
4. D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, et D. Zagorodnov, "The eucalyptus open-source cloud-computing system," in *CCGRID*, F. Cappello, C.-L. Wang, et R. Buyya, Eds. IEEE Computer Society, 2009, pp. 124–131.
5. *Open Virtualization Format Specification*, Distributed Management Task Force DMTF Standard, Rev. 1.0.0, 2009.
6. E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, et J.-B. Stefani, "The fractal component model and its support in java," *Softw., Pract. Exper.*, vol. 36, no. 11-12, pp. 1257–1284, 2006.
7. The fractal adl website. http ://fractal.ow2.org/fractaladl/
8. S. Sicard, F. Boyer, et N. D. Palma, "Using components for architecture-based management : the self-repair case," in *ICSE*, W. Schäfer, M. B. Dwyer, et V. Gruhn, Eds. ACM, 2008, pp. 101–110.
9. The usharesoft website. http ://http ://www.usharesoft.com//
10. L. Bellissard, N. D. Palma, A. Freyssinet, M. Herrmann, et S. Lacourte, "An agent platform for reliable asynchronous distributed programming," in *SRDS*, 1999, pp. 294–295.
11. *Amazon Elastic Compute Cloud User Guide*, Amazon Web Services, Nov. 2010. http ://awsdocs.s3.amazonaws.com/EC2/latest/ec2-ug.pdf
12. *Amazon Simple Storage Service Developer Guide*, Amazon Web Services, Mar. 2006. http ://awsdocs.s3.amazonaws.com/S3/latest/s3-dg.pdf
13. Microsoft azure website. http ://www.microsoft.com/windowsazure/
14. Salesforce.com website. http ://www.salesforce.com/
15. C. Chapman, W. Emmerich, F. G. Márquez, S. Clayman, et A. Galis, "Software architecture definition for on-demand cloud provisioning," in *HPDC*, S. Hariri et K. Keahey, Eds. ACM, 2010, pp. 61–72.
16. J. Mirkovic, T. Faber, P. Hsieh, G. Malayandisamu, et R. Malavia, "Dadl : Distributed application description language."
17. P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, et P. Toft, "The smartfrog configuration management framework," *SIGOPS Oper. Syst. Rev.*, vol. 43, pp. 16–25, January 2009. http ://doi.acm.org/10.1145/1496909.1496915