Département Informatique et Mathématiques Appliquées

Projet Long 2008

ADMINISTRATION AUTONOME DE SERVEURS SUR LA GRILLE AVEC UNE MACHINE VIRTUELLE

Architecture

Responsable : Daniel Hagimont - Professeur INPT/ENSEEIHT - Daniel.Hagimont@enseeiht.fr

Co-encadrant : Laurent Broto – Etudiant en thèse à l'UPS - Laurent.Broto@irit.fr

Superviseur industriel : Emmanuel Murzeau - emmanuel.murzeau@airbus.com

Chef de projet : Ezequiel Geremia - ezequiel.geremia@etu.enseeiht.fr

Etudiants :

- Julien Louisy
- Julien Clariond
- Hery Randriamanamihaga
- Ezequiel Geremia
- Mathieu Giorgino

Sommaire

1 Introduction

Nous avons à notre disposition un certain nombre d'ordinateurs que nous pouvons configurer à notre convenance. L'architecture machine que nous voulons atteindre est simple : chaque machine physique accueillera un dom0 (hyperviseur Xen permettant l'administration des machines virtuelles) et un certain nombre de machines virtuelles (appelées domU).

Le principe de la paravirtualisation Xen est le suivant : le dom0 connaît tous les domU situés sur la machine physique où il se trouve et chaque domU connaît le dom0 auquel il est rattaché. Le daemon xend qui tourne sur le dom0 lui permet d'adminstrer les domU qu'il héberge.

L'hyperviseur dom0 contient un noyau GNU/Linux modifié pour accueillir Xen et les outils d'administration (xend et autres).

Les domU sont créés à partir de dom0. Leur système de fichiers est situé sur un serveur NFS pour être accessible quelque soit la machine hôte. Ainsi, les domU conservent une référence à leur système de fichiers au cours d'une migration.

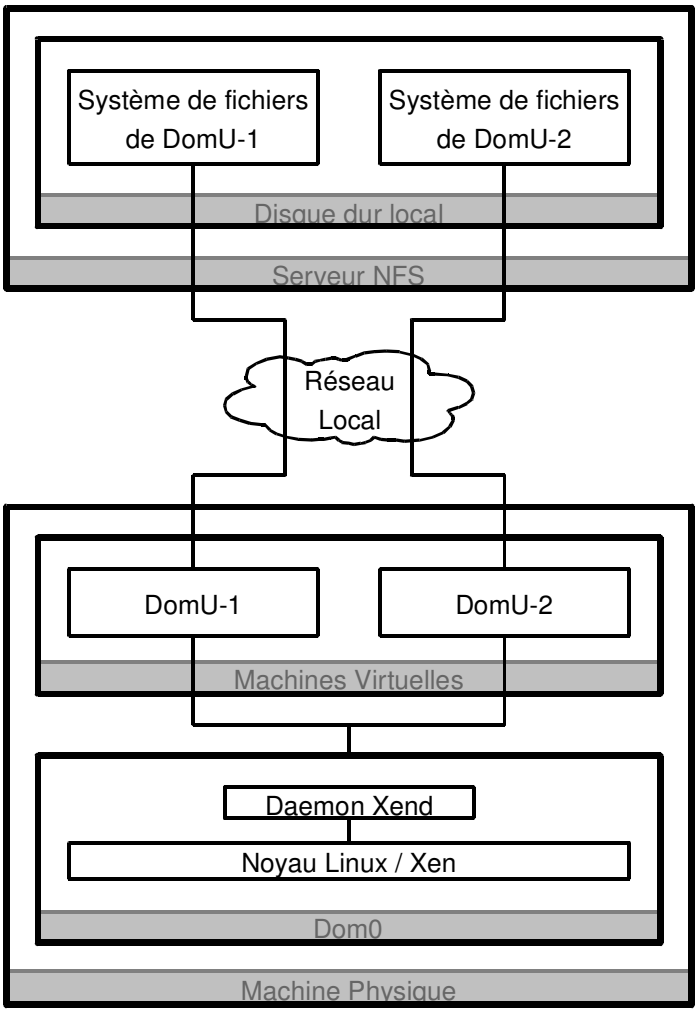

Figure 1: Architecture logicielle sur une machine physique

L'architecture machine précédente repose sur une architecture réseau spécifique. Chaque

machine virtuelle a une ou plusieurs interfaces réseau virtuelles (eth0, eth1...) qui sont reliées à des interfaces virtuelles sur le dom0 (vif1.0, vif1.1...). D'autre part, dom0 virtualise l'interface réelle de la machine physique (peth0) en une interface abstraite (eth0) à laquelle il attribue la même adresse MAC que l'interface physique. Enfin, toutes les interfaces virtuelles sont regroupées au sein d'un pont (xenbr0) qui permet aux domU d'accéder au réseau de manière transparente tout en masquant les adresses MAC aléatoires associées aux interfaces virtuelles.

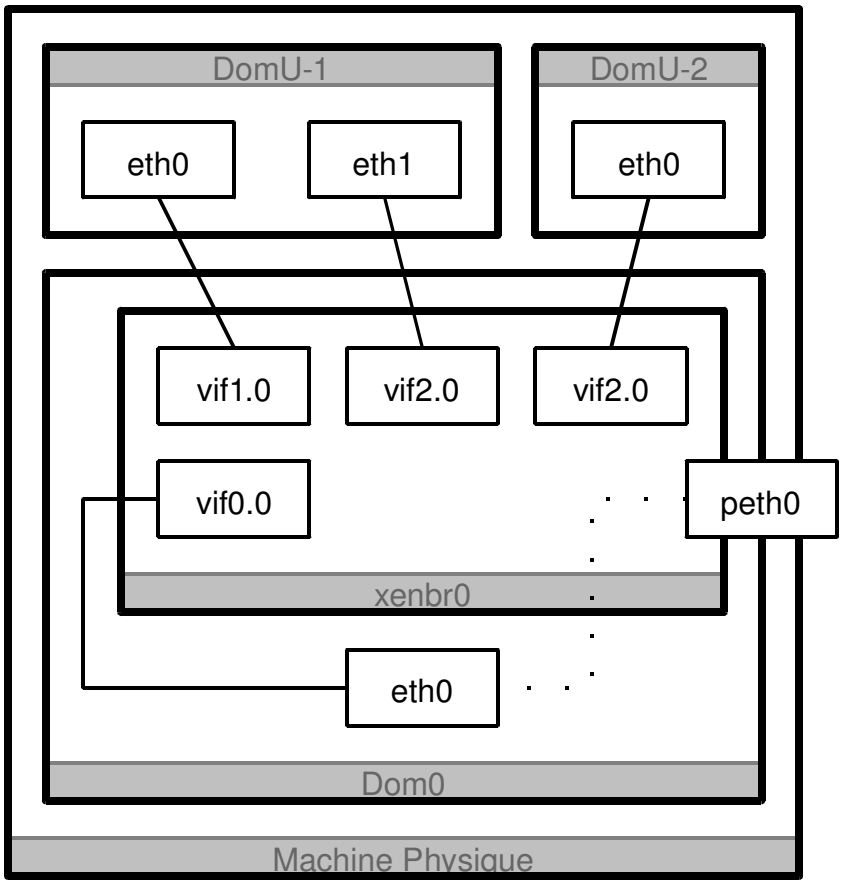

Figure 2: Architecture réseau sur une machine physique

2 Ressources matérielles

Le Centre de Compétences en Ressources Informatiques de l'ENSEEIHT a mis à notre disposition l'ensemble de ressources matérielles suivant :

- ***RM1 -*** Machine physique Dell A-204-02
  - ***RM1.1*** - Processeur : Intel(R) Pentium(R) D CPU 3.00 GHz
  - ***RM1.2*** - Mémoire : 1 Go
  - ***RM1.3*** - Carte réseau : Broadcom NetXtreme s7xx Gigabit Controller

- ***RM2 -*** Machine physique Dell A-204-03
    - ***RM2.1*** - Processeur : Intel(R) Pentium(R) D CPU 3.00 GHz
    - ***RM2.2*** - Mémoire : 1 Go
    - ***RM2.3*** - Carte réseau : Broadcom NetXtreme s7xx Gigabit Controller
- ***RM3 -*** Machine physique Dell A-204-04
    - ***RM3.1*** - Processeur : Intel(R) Pentium(R) D CPU 3.00 GHz
    - ***RM3.2*** - Mémoire : 1 Go
    - ***RM3.3*** - Carte réseau : Broadcom NetXtreme s7xx Gigabit Controller
- ***RM4 -*** Machine physique Dell A-204-05
    - ***RM4.1*** - Processeur : Intel(R) Pentium(R) D CPU 3.00 GHz
    - ***RM4.2*** - Mémoire : 1 Go
    - ***RM4.3*** - Carte réseau : Broadcom NetXtreme s7xx Gigabit Controller
- ***RM5 -*** Machine physique Dell A-204-06
    - ***RM5.1*** - Processeur : Intel(R) Pentium(R) D CPU 3.00 GHz
    - ***RM5.2*** - Mémoire : 1 Go
    - ***RM5.3*** - Carte réseau : Broadcom NetXtreme s7xx Gigabit Controller
- ***RM6*** – Hub
- ***RM7*** – Câbles RJ45

3 Ressources Logicielles

Les systèmes d'exploitation installés sur les Dom0 des machines du parc sont des Ubuntu Gutsy Gibbon, choisis pour leur facilité de déploiement et nos connaissances préalables de leur utilisation. Gutsy Gibbon permet, par ailleurs, de mettre en place sans trop de difficulté Xen par

l'intermédiaire du gestionnaire de paquets Synaptics.

La distribution Ubuntu Gutsy Gibbon n'étant pas stable pour une utilisation en tant que DomU, les systèmes d'exploitation choisis pour les machines virtuelles sont Ubuntu Dapper Drake et Debian GNU/Linux Etch. Les machines virtuelles nécessitent de démarrer sur NFS. Cette fonctionnalité ne peut être activée qu'au moment de la compilation de leur noyau. Les sources du noyau xen-linux 2.6.18.8 n'étant pas accessibles, nous avons dû compiler les sources du noyau xen-linux 2.6.18.8 pour une utilisation sur les DomU.

Nous avons opté pour un hébergement du serveur de contrôle de version par Google Code, au détriment d'un serveur hébergé par nos soins, afin de minimiser les coûts de déploiement, d'accès et de maintenance.

Les logiciels qui nous ont été imposés par le client sont les suivants:

- TUNe
- Java pour l'utilisation, entre autres, de RMI
- Xen
- NFS en tant que système de fichiers des domU

Enfin, les systèmes d'exploitation et les logiciels utilisés lors du projet sont les suivants :

- ***RL1*** - Systèmes d'exploitations
  - ***RL1.1*** - Ubuntu Gutsy Gibbon (7.10) sur machine physique
  - ***RL1.2*** - Ubuntu Dapper Drake (6.06) sur machine virtuelle
  - ***RL1.3*** - Debian GNU/Linux Etch (4.0R2) sur machine virtuelle
- ***RL2*** - Outil de virtualisation
  - ***RL2.1*** - Xen 3.1
  - ***RL2.2*** - Noyau xen-linux 2.6.18
- ***RL3*** - Outils complémentaires
  - ***RL3.1*** - Outil de configuration des ponts Ethernet Linux - *bridge-utils 1.2-1build1*
  - ***RL3.2*** - Outils professionnels de contrôle du réseau dans les noyaux Linux - *iproute 20070313ubuntu2*

- ***RL3.3*** - Gestionnaire de volumes logiques Linux - *lvm2 2.02.26-ubuntu4*
- ***RL3.4*** - Analyseur de trafic réseau - *wireshark 0.99.6rel-3ubuntu0.1*
- ***RL3.5*** - Tune - *version fournie par le client*
- ***RL3.6*** - Système de contrôle de versions avancé - *Subversion hébergé par Google Code*
- ***RL3.7*** - Système de fichier en réseau - *NFS (Network File System)*
- ***RL3.8*** - Java SE Development Kit 6

4 Structure du parc

Les machines virtuelles nécessitent un système de fichier délocalisé et accessible quelle que soit leur localisation sur les machines physiques et sur le réseau. L'utilisation d'un système à base de serveur NFS nous a été imposé par le client pour répondre à ce besoin. Nous avons opté pour un serveur NFS unique, afin de concentrer les services et éviter ainsi une multiplication inutile de la maintenance. Bien que l'espace disque utilisable sur le parc se retrouve ainsi réduit à l'espace disponible uniquement sur le serveur, cette situation s'avère peu handicapante dans la mesure où cet espace est suffisant pour le nombre de machines virtuelles envisagé. Bien entendu, la sécurité d'un tel système est assuré par la paramétrisation du serveur NFS qui permet de spécifier les zones accessibles par les machines virtuelles ainsi que leurs privilèges d'utilisation (accès en écriture/lecture, ...). Néanmoins, ces espaces de stockage sont partagés entre toutes les machines virtuelles, chacune pouvant accéder aux données des autres.

Afin d'élargir notre champ d'actions et de nous affranchir des limitations imposées par le Centre de Compétences en Ressources Informatiques de l'école, nous avons mis en place un réseau local indépendant. Pour cela, nous avons configuré un serveur DHCP et opté pour l'utilisation d'un switch.

Tune désignant les machines par leurs noms et non par leurs adresses IP, l'utilisation d'un système de noms de domaine s'est avérée nécessaire. Ainsi, nous avons dû configurer un serveur DNS pour mettre en place un service de désignation de plus haut niveau.

Pour assurer la synchronisation temporelle des Dom0, un serveur NTP a été ajouté à l'architecture, configuré pour diffuser l'heure aux machines du réseau mais n'acceptant aucune de leurs requêtes.

Enfin, dans le but de centraliser au maximum les services et ainsi de minimiser la maintenance, le serveur NFS, le serveur DHCP, le serveur DNS et le serveur NTP ont été regroupés sur une même machine physique dédiée.

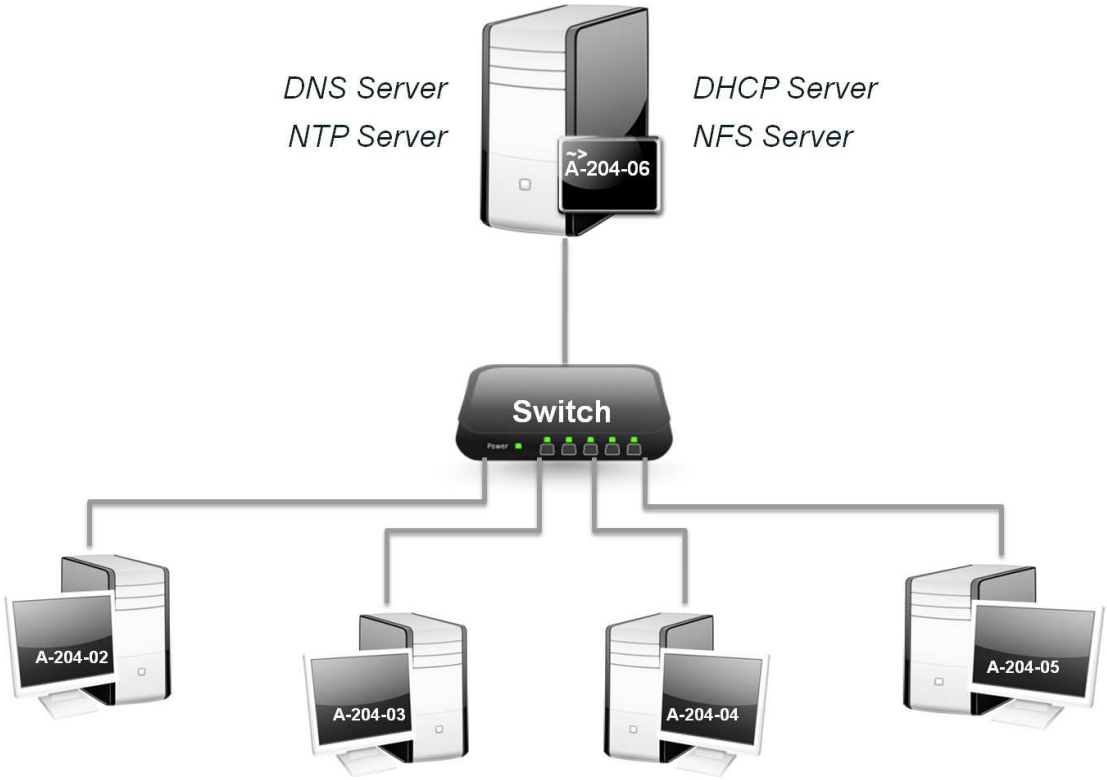

Figure 3: Structure du parc

4.1 Serveur

- Système d'exploitation : GNU/Linux Ubuntu Gutsy Gibbon (7.10)
- Serveur DHCP – 3.0.5-3
- Serveur NFS :
  - Version: nfs-kernel-server – 1:1.1.1~git200707
  - Exporte une partition de 70 Go qui contient les systèmes de fichiers des machines virtuelles

4.2 Machines clientes

- Système d'exploitation : GNU/Linux Ubuntu Gutsy Gibbon (7.10)

4.3 Machines virtuelles

- Systèmes d'exploitations : GNU/Linux Debian Etch (4.02r2)
- Système de fichier hébergé sur la machine Serveur

5 Architecture autonome administrée par Tune

L'architecture TUNe mise en place repose sur une encapsulation des domaines utilisateurs dans leur globalité ainsi que sur la mise en place de sondes qui initient la migration en réponse à un stimulus prédéfini (ici, la charge CPU).

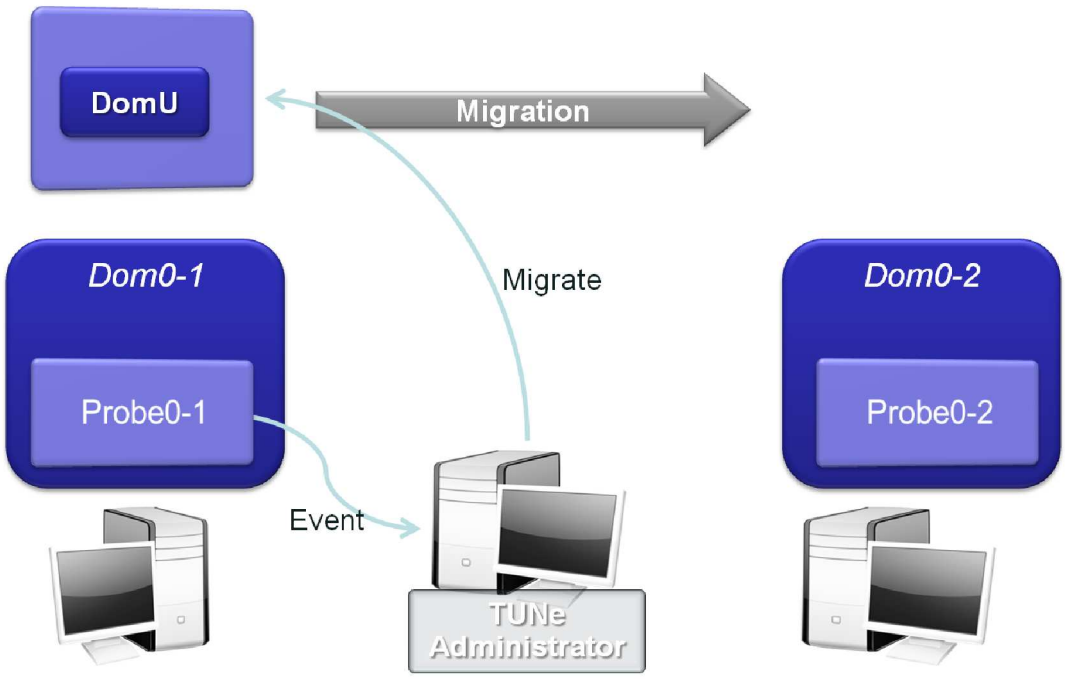

Figure 4: Architecture globale de l'encapsulation dans TUNe

Nos wrappers sont relativement simples :

- Les domaines utilisateurs sont encapsulés dans des composants présentant les stubs suivants :
  - start qui démarre la machine virtuelle
  - stop qui l'arrête (équivalent d'un `#sudo halt`)
  - migrate qui initie une migration
- Les sondes sont encore plus simples : uniquement deux interfaces pour respectivement démarrer et éteindre la sonde.

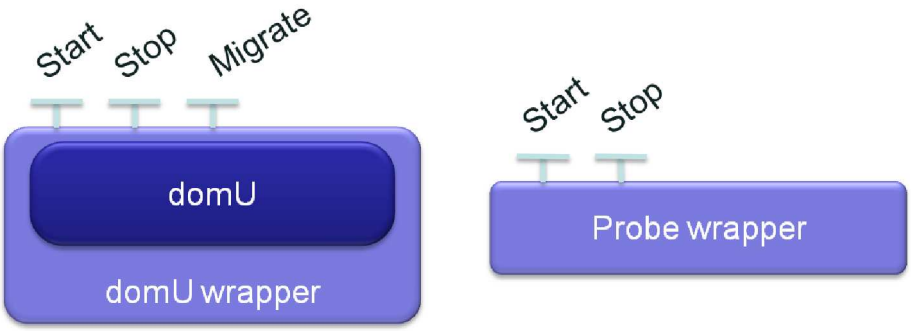

Figure 5: Wrappers

Le diagramme de déploiement (Figure 8) montre deux aspects propres à notre projet :

- TUNe exige le déploiement d'une archive contenant le logiciel patrimonial, dont nous pouvons nous abstraire. En effet, les domU s'administrant directement à partir des dom0 présents sur les machines physiques, nous utilisons le fichier fictif fake.tar.gz. Dans une version plus correcte des wrappers, il serait judicieux d'intégrer à cette archive les exécutables de notre application.
- Les fichiers XML de description des wrappers, utilisant le langage WDL (Wrapper Description Language), sont très simples :
  - Pour les domUs : on utilise les méthodes définies dans la classe VM avec le composant encapsulant le domU en paramètre. Pour la méthode migrate, le deuxième argument est le nom de la machine physique destination.

```
<wrapper name="Xen">
      <method name="start" key="xen.VM" method="start">
            <param value="$this.srname"/>
      </method>
      <method name="stop" key="xen.VM" method="shutdown">
            <param value="$this.srname"/>
      </method>
      <method name="migrate" key="xen.VM" method="migrate">
            <param value="$this.srname"/>
            <param value="a-204-05"/>
      </method>
</wrapper>
```

Figure 6: Fichier WDL d'une machine virtuelle

  - Pour les sondes : on appelle les méthodes définies dans la classe Probe. La méthode create permet de lancer la sonde. On lui fournit en arguments le *pipe* dans lequel écrire les évènements, une référence au composant auquel est attachée la sonde et le seuil CPU à partir duquel la machine virtuelle observée migre (ici 100%).

```
<wrapper name="XenProbe">
      <method name="start" key="xen.Probe" method="create">
            <param value="$tubeAddr"/>
            <param value="$probed.srname"/>
            <param value="1"/>
      </method>
      <method name="terminate" key="xen.Probe" method="finalize">
      </method>
</wrapper>
```

Figure 7: Fichier WDL de la sonde développée

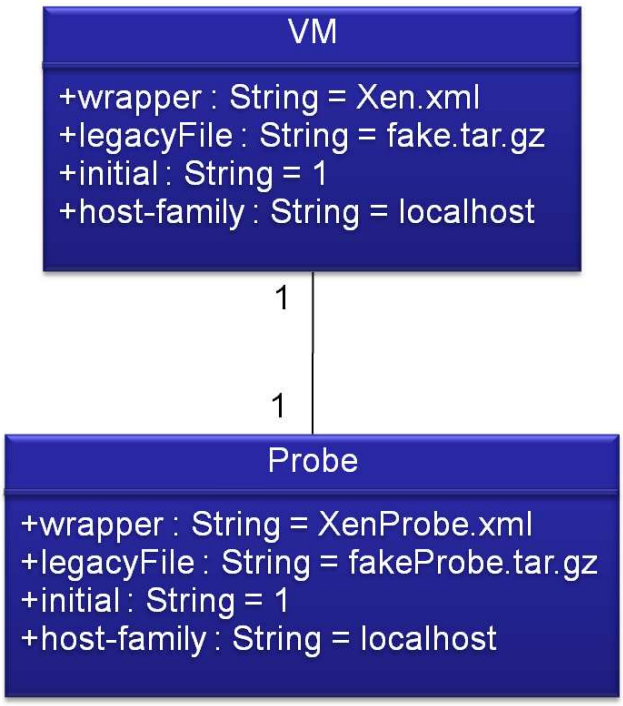

Figure 8: Diagramme de Déploiement

Les diagrammes de reconfiguration ne sont pas beaucoup plus complexes que les wrappers. Les diagrammes répondant aux évènements start et stop permettent de démarrer/arrêter successivement domU et sonde. Pour la migration, on migre la machine virtuelle liée à la sonde.

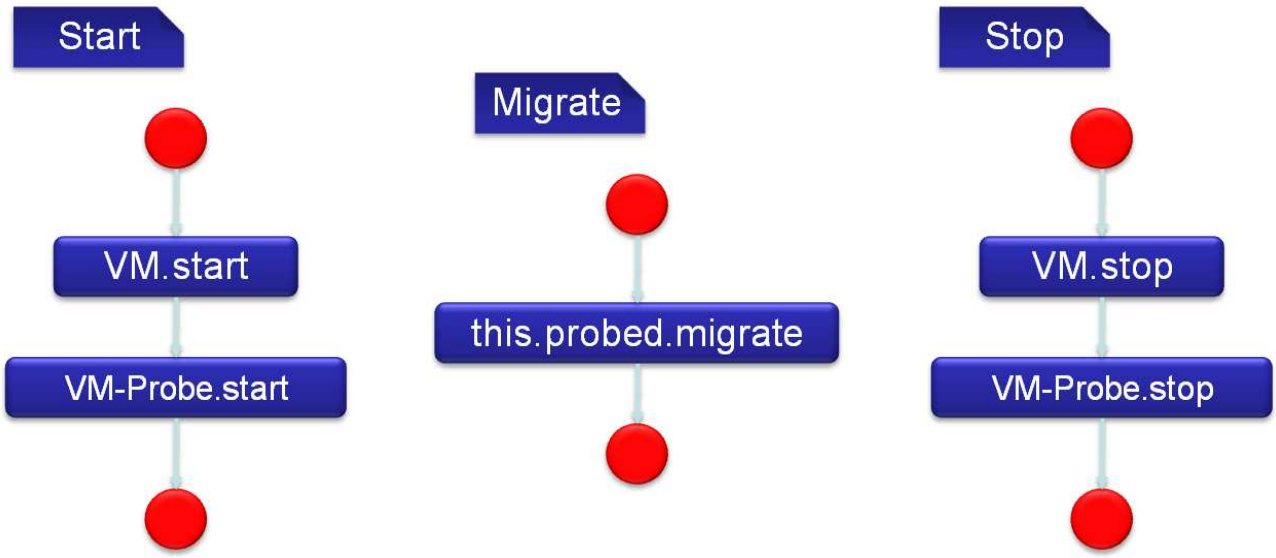

Figure 9: Diagrammes de Reconfiguration