Département Informatique et Mathématiques Appliquées

Projet Long 2008

ADMINISTRATION AUTONOME DE SERVEURS SUR LA GRILLE AVEC UNE MACHINE VIRTUELLE

Rapport d'activités

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 Description

1.1 Contexte

La virtualisation a pour but de permettre l'exécution de plusieurs systèmes d'exploitation sur une même machine physique. Le système d'exploitation installé sur la machine physique est appelé hôte. Celui-ci émule des machines qualifiées de virtuelles. Les machines virtuelles peuvent à leur tour accueillir des systèmes d'exploitation dans le but d'opérer des traitements particuliers.

Le cadre d'utilisation des outils de virtualisation est illustré dans l'exemple qui suit. Plus généralement, il s'agira de minimiser un certain critère (mesuré selon une métrique bien définie) grâce au mécanisme de migration. La migration consiste à déplacer un composant logiciel de la machine physique sur laquelle il s'exécute à une autre à travers un réseau (cf. Figure 1.1.1). Si le composant en question est un système d'exploitation, alors il est déplacé avec son contexte d'exécution, mais sans son système de fichiers.

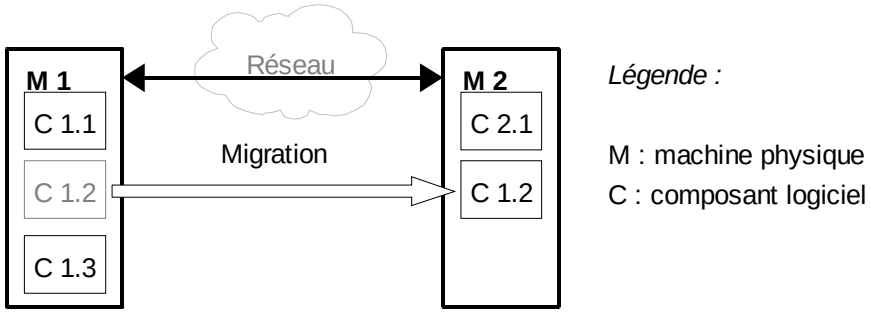


Figure 1.1.1: Schéma illustrant le principe de la migration

Les grilles d'ordinateurs sont capables de traiter un volume important d'information correspondant à des calculs de simulation ou à des requêtes de clients vers des serveurs Web. Certains hébergeurs proposent donc de louer des sous-ensembles de ces grilles. L'hébergeur doit donc faire face à la problématique suivante : comment maximiser le gain financier issu de la location de serveurs ?

Pour cela, il peut diminuer le coût d'exploitation de la grille, assimilé ici au coût de la consommation électrique. Cela se traduit notamment par l'extinction des machines physiques faiblement chargées, i.e. dont la charge processeur est sous un certain seuil minimum. Cependant, l'application qui s'exécute sur la machine à éteindre doit être déplacée avec son contexte d'exécution vers un autre serveur qui peut la supporter. Si l'ensemble des applications n'est plus supporté par les machines physiques allumées, alors d'autres machines physiques peuvent être mises sous tension afin d'accueillir les applications à forte consommation processeur.

L'hébergeur peut également maximiser le taux d'occupation de la grille, i.e. maximiser la charge processeur sur l'ensemble des machines physiques. Il s'agit, d'une part, de concentrer les applications à faible consommation CPU sur un nombre réduit de machines physiques et d'autre part d'allouer dynamiquement des machines physiques à des applications fortement consommatrices en

CPU. Ceci permet d'augmenter le nombre de machines physiques disponibles et, de ce fait, d'augmenter la capacité d'accueil de nouvelles applications. Il en résulte qu'il est possible de louer plus de machines physiques qu'il n'en existe sur la grille, tout en assurant un fonctionnement global optimal. En effet, la période de pointe, en terme de volume d'information à traiter, pour certaines applications, coïncide avec la période creuse pour d'autres et vice versa.

Enfin, si la grille est répartie sur des sites spatialement distants, alors le déplacement d'applications qui communiquent entre-elles sur des points plus proches du réseau est une solution qui diminue le temps de communication inter-applicatif.

Ces trois politiques, résumées dans la Figure 1.1.2, peuvent être mises en oeuvre grâce aux outils de virtualisation.

Contexte	parc de serveurs payants	parc de serveurs à haute consommation électrique	parc de serveurs distribué sur des sites distants
Métrique	prix du temps d'occupation par serveur et par seconde	ampérage électrique de l'alimentation par serveur	distance de la route séparant deux serveurs et charge réseau
But	minimiser le coût = concentrer les tâches à faible charge processeur	minimiser la consommation électrique = éteindre les machines physiques à faible charge processeur (sans perte de tâche)	minimiser le temps de communication = regrouper les tâches sur un réseau local

Figure 1.1.2: Tableau récapitulatif de la politique à mettre en oeuvre en fonction du contexte

Afin d'effectuer des mouvements de serveurs, ces derniers sont exécutés sur des machines virtuelles sous un système d'exploitation quelconque. De cette manière, il est possible d'administrer des machines virtuelles hétérogènes. Nous utiliserons l'outil de paravirtualisation Xen qui permet d'effectuer des migrations à chaud i. e. sans interruption de service.

Afin d'automatiser l'administration de machines virtuelles, nous utiliserons l'outil TUNe qui autorise une administration autonome de haut niveau. TUNe utilise les composants Fractal pour fournir une interface homogène quelle que soit la machine virtuelle encapsulée.

1.2 Objectifs

Les trois principaux objectifs de ce projet sont :

- La maîtrise de la technologie de paravirtualisation Xen.
- Son intégration dans le gestionnaire de composants TUNe en vue d'une administration autonome.
- L'implémentation d'une politique de distribution des machines virtuelles en fonction de la charge processeur.

1.2.1 Maîtrise de Xen et de son mécanisme de migration

L'administration de serveurs doit s'effectuer de manière transparente pour le client, ainsi :

- Les connexions TCP des machines virtuelles doivent être maintenues lors d'une migration.
- La migration d'une machine virtuelle ne doit pas affecter le résultat des applications qui s'y exécutent.

L'utilisation de l'outil de paravirtualisation Xen nous a été imposé car il autorise des migrations de machines virtuelles respectant les exigences précédentes. En particulier :

- La migration de machines virtuelles est effectuée sans interruption d'activité.
- La migration de machines virtuelles peut être effectuée quelle que soit la charge processeur de la machine physique contenant le système d'exploitation hôte.

Évidemment, ces propriétés sont assurées par Xen mais elles devront être vérifiées par des mesures de performances. Dans cette optique, seront quantifiés :

- La durée de la migration d'une machine virtuelle.
- Le délai introduit par la migration.
- La durée de l'interruption de service lors d'une migration.

1.2.2 Intégration dans TUNe

TUNe permet l'administration de haut niveau de composants de type Fractal. Afin d'intégrer l'architecture associée au projet dans TUNe, chaque système d'exploitation s'exécutant sur les machines virtuelles devra être encapsulé dans un composant Fractal.

L'intégration dans TUNe ne doit pas nuire aux performances. Ainsi, les propriétés obtenues précédemment devront être préservées :

- Les connexions TCP à des machines virtuelles seront maintenues lors d'une migration.
- La migration d'une machine virtuelle n'affectera pas le résultat des applications qui s'y exécutent.
- La migration de machines virtuelles sera effectuée quelle que soit la charge processeur de la machine hôte.

1.2.3 Administration autonome

TUNe permet l'intégration de sondes décidant des actions à effectuer. Ceci doit être utilisé afin de respecter les objectifs suivants :

- Dans l'environnement TUNe, le déclenchement de la migration devra être automatique.
- La métrique de déclenchement de migration sera le taux d'utilisation CPU.
- La migration sera initiée par le dépassement d'un seuil CPU fixé.

1.2.4 Contraintes

Les outils à utiliser seront :

- Des systèmes d'exploitations de type GNU/Linux sur les machines physiques.
- L'outil de virtualisation Xen.
- TUNe en tant que gestionnaire de composants.

L'architecture réseau devra respecter les critères suivants :

- Les machines physiques seront connectées par un réseau
- Le protocole NFS (Network File System) sera utilisé pour l'échange de fichiers sur le réseau.

Le projet devra satisfaire les contraintes de mise en oeuvre ci-après :

- Les machines virtuelles devront être encapsulées dans des composants de type Fractal.
- Un diagramme de reconfiguration, propre à l'environnement, contrôlera la migration de machines virtuelles.
- Dans l'environnement TUNe, une sonde sera développée pour prélever la charge processeur des machines physiques sur lesquelles s'exécutent les systèmes d'exploitation hôtes.

2 Technologies utilisées

2.1 Xen

2.1.1 Présentation du projet Xen

Xen est un projet de virtualisation par hyperviseur fondé en 2003 par Ian PRATT. La solution de virtualisation Xen est séparée en plusieurs produits ayant différentes fonctionnalités. La version libre concentre toute la technologie de virtualisation. Les autres versions propriétaires de la gamme se distinguent uniquement par le support proposé, le nombre de machines virtuelles supportées, les systèmes invités supportés ainsi que les logiciels annexes, la technologie de virtualisation étant celle utilisée par la version libre de Xen.

Dans sa première version, Xen restait limité dans ses performances et ses fonctionnalités. Actuellement, le projet en est à la version 3.2 qui propose de meilleurs performances ainsi qu'un meilleur support des instructions de virtualisation.

Xen est un hyperviseur, c'est-à-dire qu'il vient s'insérer directement entre le matériel et le noyau. C'est donc Xen qui a l'accès exclusif au matériel, et les systèmes d'exploitations instanciés par dessus doivent passer par l'hyperviseur pour y accéder. Par ailleurs, l'utilisation de Xen nécessite la modification des couches basses des systèmes invités pour, d'une part permettre la cohabitation des systèmes portés avec l'hyperviseur, et d'autre part, améliorer les performances.

Au dessus de l'hyperviseur se trouvent les systèmes invités, contrôlés par Xen. Le premier système démarré par Xen, appelé « domaine zéro » (Dom0), par opposition aux systèmes démarrés plus tard, les « domaines utilisateurs » (DomU), a des privilèges particuliers. Seul le domaine zéro a un accès complet au matériel, à travers l'hyperviseur et peut communiquer directement avec lui pour instancier les domaines utilisateurs. C'est donc ce dernier qui effectue la configuration des systèmes invités : la création des fichiers de configuration des domaines utilisateurs, la réservation de l'espace disque, l'allocation de la mémoire. Inversement, les domaines utilisateurs ont uniquement accès à ce que l'administrateur a configuré, à savoir, les partitions physiques exportées par l'hyperviseur et éventuellement les lecteurs partagés sur le réseau.

Enfin, le projet Xen comprend un ensemble de modifications à appliquer au noyau Linux ainsi que des utilitaires permettant d'administrer les domaines utilisateurs. Ces utilitaires sont installés dans l'espace utilisateur du domaine 0 et interagissent avec l'hyperviseur.

La Figure 2.1.1 synthétise cette séparation modulaire du projet Xen, dans laquelle on distingue l'hyperviseur, l'intégration de l'interface de contrôle au noyau Linux ainsi que les programmes de contrôle en espace utilisateur.

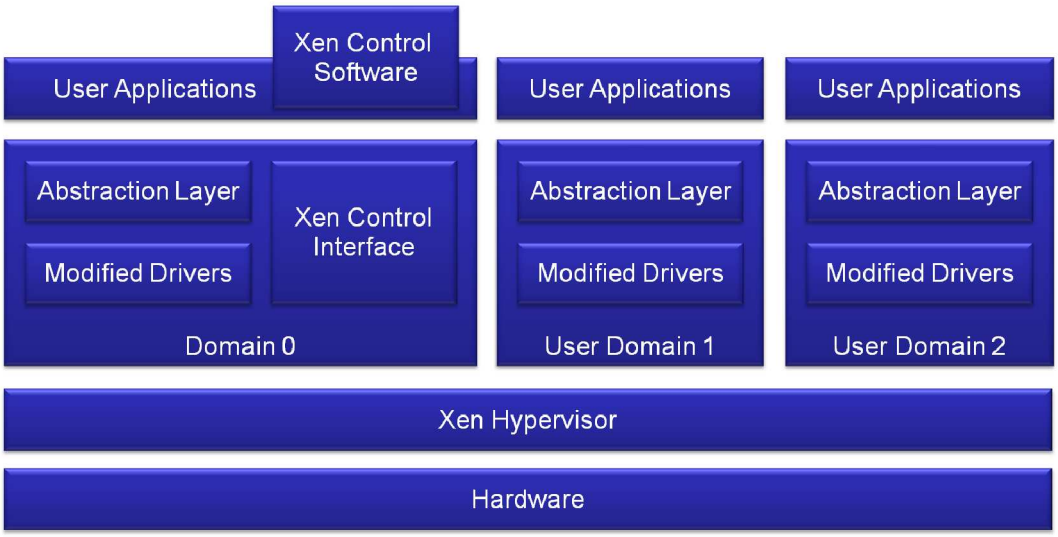


Figure 2.1.1: Architecture de Xen

2.1.2 L'hyperviseur Xen

L’hyperviseur est démarré directement par le chargeur de démarrage de l’ordinateur, en lieu et place d’un système d’exploitation traditionnel. L’hyperviseur instancie ensuite le domaine 0 qui réalise seul le reste de l’initialisation des périphériques.

En pratique, au démarrage de la machine, on voit s’afficher quelques lignes traitant de l’hyperviseur et de sa configuration avant que le noyau Linux ne prenne le relais et entame le processus de démarrage « traditionnel ».

Un accès en lecture à des fichiers contenant les informations de configuration de l'hyperviseur est impossible au démarrage du système car le système de fichiers n’est pas encore activé au moment où l’information est nécessaire. Ainsi, comme pour le noyau Linux, le chargeur de démarrage fournit une partie de la configuration de l’hyperviseur. Elle comprend des éléments de configuration non modifiables en cours d’exécution comme la quantité de RAM à affecter au domaine 0, les paramètres de la console série, ... Dans la configuration du chargeur de démarrage, l’hyperviseur apparaît comme un système d’exploitation de type GNU/Linux, avec simplement des options différentes de celles passées à un noyau Linux.

Techniquement, l’hyperviseur est le point de passage obligatoire pour tout accès au matériel. Il régule et répartit les accès aux ressources entre les systèmes invités (domaine 0 et domaines utilisateurs).

Dans Xen, la mémoire physique installée est répartie sans recouvrement entre les systèmes invités. C’est à dire que si la machine dispose de 512 mégaoctets de RAM, l’administrateur pourra par exemple faire fonctionner 4 systèmes invités (incluant le domaine 0) disposant chacun de 128 mégaoctets. Il ne pourra pas allouer plus de mémoire qu’il n’y en a de disponible, la mémoire n’est pas une ressource « partageable » entre les systèmes d’exploitation. Il est par contre tout à fait possible d’arrêter un domaine utilisateur et d’en démarrer un autre à la place, du moment que le nouveau système ne nécessite pas plus de RAM qu’il n’en reste.

À l’inverse, l’accès au processeur n’est pas exclusif, tous les systèmes invités en cours d’exécution ont chacun une part du temps processeur total disponible sur la machine.

Ensuite, l'accès au matériel réseau repose sur une abstraction réseau spécifique. Chaque machine virtuelle a une ou plusieurs interfaces réseau virtuelles (eth0, eth1...) qui sont reliées à des interfaces virtuelles sur le domaine 0 (vif1.0, vif1.1...). D'autre part, le domaine 0 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 domaines utilisateurs d'accéder au réseau de manière transparente tout en masquant les adresses MAC aléatoires associées aux interfaces virtuelles.

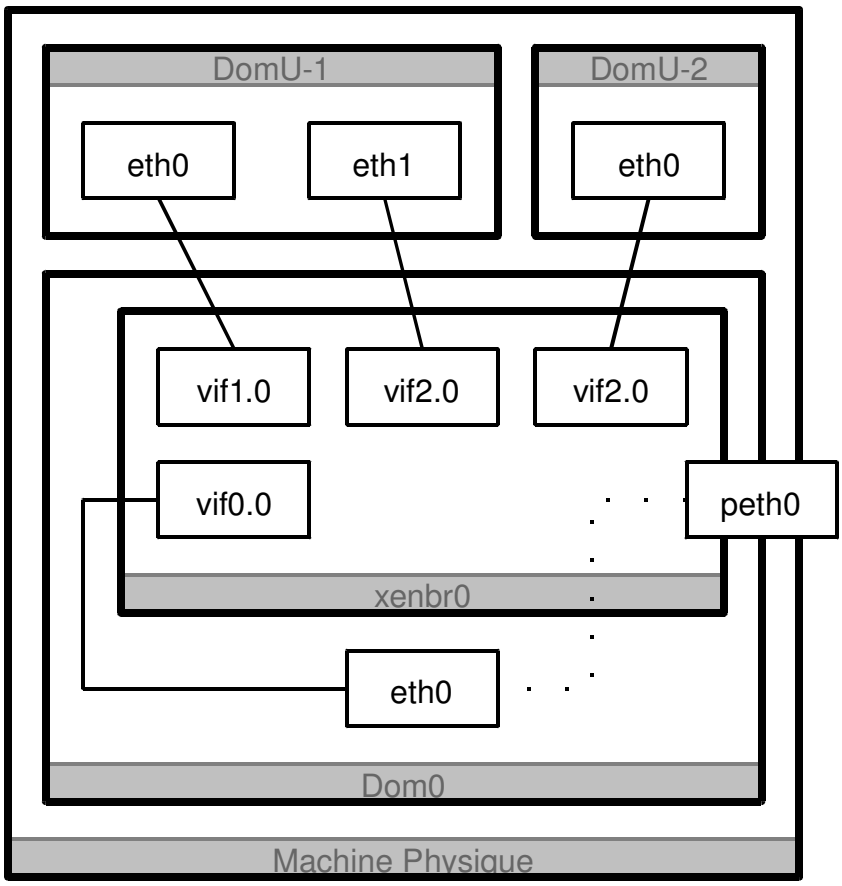


Figure 2.1.2: Architecture réseau sur une machine physique

En plus de gérer l'accès aux ressources, l'hyperviseur doit aussi gérer la correspondance entre la représentation de la mémoire des systèmes invités et la disposition effective de la mémoire pour le processeur. De manière simplifiée, chaque système invité gère lui-même sa mémoire comme s'il était seul, dans un espace mémoire réservé, et tient au courant l'hyperviseur des opérations relatives à la gestion de la mémoire. L'hyperviseur est lui en charge de faire la correspondance entre l'adresse virtuelle du système invité et l'adresse réelle manipulée par le processeur.

Le noyau Linux et l'hyperviseur communiquent par le biais d'appels systèmes spécifiques, appelés *hypercalls* (pour hypervisor calls, appels à l'hyperviseur). Ces appels permettent de passer des messages à l'hyperviseur, sur les tâches à accomplir (par exemple l'allocation ou la libération de mémoire).

2.1.3 Modifications apportées au noyau Linux

La plupart des modifications apportées au noyau Linux portent sur l'ajout de paravirtualisation pour la coopération avec l'hyperviseur, notamment pour la gestion de la mémoire, qui concentre la majorité des modifications.

Ces patches ont pour effet de rajouter une architecture matérielle dans les options de compilation du noyau xen_x86. Cette architecture matérielle, « fictive » car elle ne correspond pas réellement à du matériel, permet de regrouper toutes les modifications, au lieu de modifier directement les fichiers concernés pour l'architecture x86 (opération très intrusive).

La communication avec l'hyperviseur s'effectue au moyen d'hypercalls. Il y a en tout une trentaine d'hypercalls définis, couvrant toutes les opérations courantes de Xen : gestion de la mémoire, gestion des Entrées/Sorties, administration des systèmes virtualisés, etc.

Si la communication depuis le système invité vers l'hyperviseur est réalisée par un hypercall, le passage d'informations dans le sens inverse est réalisé par un bus d'évènements au fonctionnement très similaire à celui des interruptions matérielles. Dès qu'un évènement susceptible de nécessiter une action de la part du système survient (arrivée d'un paquet sur l'interface réseau, fin d'un transfert de données sur le disque, ...), il est signalé au système, qui prend alors le relais et traite l'évènement. Les bus d'évènements de Xen ont le même rôle, mais pour la transmission d'informations relatives à l'hyperviseur (par exemple domaine utilisateur créé ou arrêté avec succès).

Les modifications apportées couvrent donc la gestion des messages de l'hyperviseur Xen, les hypercalls et l'adaptation du noyau à l'hyperviseur.

2.1.4 Programmes de contrôle

Les applications utilisateurs (programmes permettant de contrôler l'exécution des domaines utilisateurs) se situent exclusivement dans le domaine 0.

L'application principale de l'espace utilisateur est xend (pour Xen daemon). xend est démarré en tant que service avec le système d'exploitation et sert d'interface entre les hypercalls de l'hyperviseur et les outils de contrôle de Xen. Il est aussi chargé de faire passer les informations du bus d'évènement vers l'espace utilisateur.

La Figure 2.1.3 représente l'architecture de Xen. Les applications utilisateurs communiquent avec xend selon le modèle client/serveur, xend assurant le rôle du serveur. La communication avec le noyau s'effectue sous la forme d'appels systèmes traditionnels Unix. Le noyau fait alors un hypercall en fonction de l'appel système et des paramètres reçus.

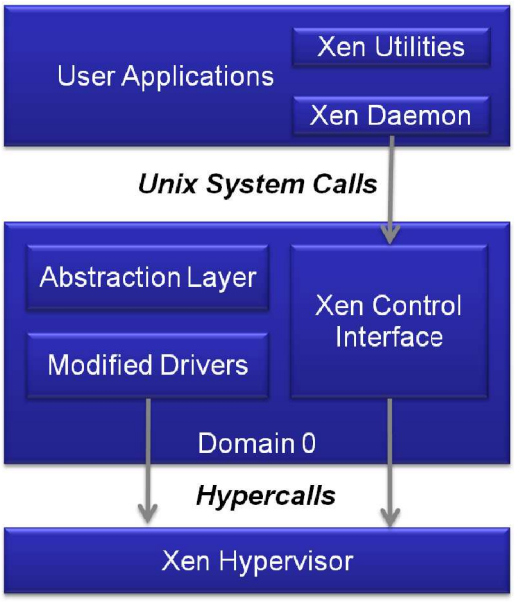


Figure 2.1.3: Enchaînement des appels systèmes au sein de Xen

Les autres outils de gestion sont principalement des commandes servant d'interface à xend. Ils permettent de démarrer, de migrer, d'arrêter les domaines utilisateurs ou encore de surveiller leur état (notamment la consommation mémoire et le temps processeur), un peu à la manière de la commande top de GNU/Linux.

2.2 Le système autonome TUNe

2.2.1 Présentation de TUNe

Les environnements informatiques d'aujourd'hui sont de plus en plus sophistiqués. Ils intègrent de nombreux logiciels complexes qui coopèrent dans le cadre d'une infrastructure logicielle, potentiellement à grande échelle. Ces logiciels se caractérisent par une grande hétérogénéité, en particulier en ce qui concerne leurs fonctions d'administration qui sont le plus souvent propriétaires.

Par conséquent l'administration de ces infrastructures logicielles (installation, configuration, réparation, optimisation ...) est une tâche très complexe, source d'erreurs, et consommatrice en ressources humaines.

Une approche très prometteuse consiste à implanter un logiciel d'administration autonome. Ce logiciel fournit un support pour le déploiement et la configuration des applications dans un environnement réparti. Il fournit également un support pour la supervision de l'environnement administré et permet de définir des réactions face à des événements comme des pannes ou des surcharges, afin de reconfigurer les applications administrées de façon autonome.

TUNe est un exemple de tel système.

2.2.2 Une architecture à composants

TUNe s'appuie sur un modèle à composants afin de fournir une vision uniforme d'un environnement logiciel composé de différents types de logiciels. Chaque logiciel administré est encapsulé dans un composant et l'environnement logiciel est abstrait sous la forme d'une architecture à composants (voir Figure 2.2.1).

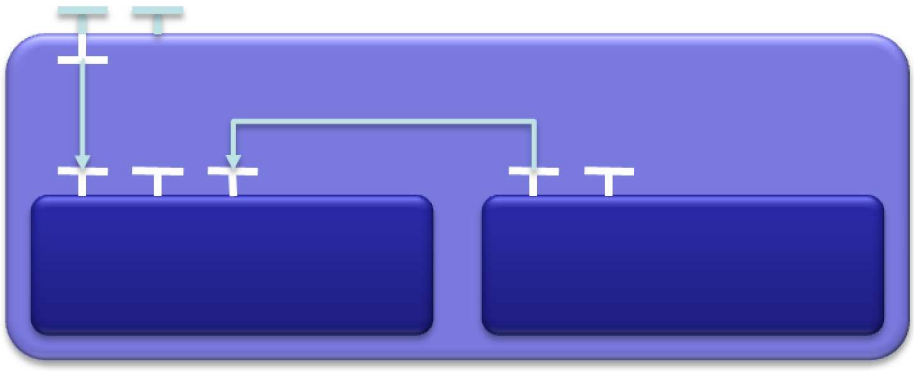


Figure 2.2.1: Principe d'un modèle à composants.
L'architecture peut être considérée comme un ensemble de composants incluant éventuellement d'autres composants, et offrant des interfaces de pilotage.

Ainsi, le déploiement, la configuration et la reconfiguration de l'environnement logiciel peuvent être réalisés en utilisant les outils associés à ce modèle. Plus précisément, TUNe utilise le modèle à composants Fractal.

Tout logiciel patrimonial (Legacy Software) géré par TUNe est encapsulé dans un composant Fractal (que nous appelons Wrapper), qui fournit une interface permettant son administration locale (voir).

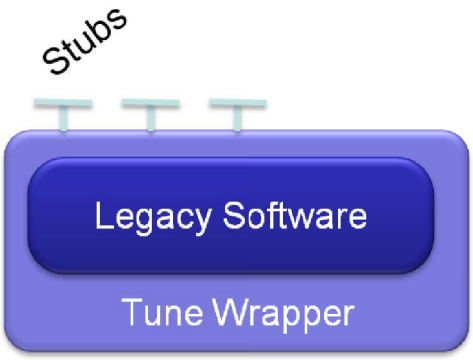


Figure 2.2.2: Principe d'un wrapper TUNe

Le logiciel patrimonial est entièrement wrappé et interfacé par des stubs définis.

Ainsi, le modèle à composants Fractal est utilisé pour implanter une couche d'administration au dessus de la couche patrimoniale (composée des logiciels administrés). Dans cette couche, les composants fournissent une interface d'administration pour les logiciels encapsulés dont le comportement est spécifique au logiciel (défini par le wrapper associé). Cette interface permet ainsi de contrôler l'état du composant de manière homogène, en évitant des interfaces de configuration complexes et propriétaires. Enfin, les interfaces de contrôle fournies par Fractal permettent d'administrer les attributs des composants et de relier ces derniers entre eux.

2.2.3 Principe d'utilisation : encapsulation, déploiement et reconfiguration

L'interface d'un modèle à composants comme Fractal est, cependant, de trop bas niveau. Plus précisément, l'utilisation du modèle à composants ne simplifie pas les points suivants :

- Les wrappers doivent utiliser l'interface de programmation du modèle à composant Fractal. L'approche de TUNe pour résoudre ce problème consiste à introduire un Langage de Description de Wrapper (WDL) permettant de décrire leur comportement (voir Figure 2.2.3). A l'exécution, une description WDL est interprétée par un wrapper générique (un composant Fractal) et évite ainsi au développeur de manipuler l'ADL Fractal.

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<wrapper name='S1'>

      <method name="start" key="package.S1" method="start" >
            <param value="$this.srname"/>
      </method>

      <method name="stop" key="package.S1" method="shutdown" >
            <param value="$this.srname"/>
      </method>

      <method name="configure" key="package.S1" method="configure" >
            <param value="$this.srname"/>
            <param value="conf.file"/>
      </method>

      <method name="reconfigure" key="package.S1" method="reconfigure" >
            <param value="$this.srname"/>
      </method>

</wrapper>
```

Figure 2.2.3: Exemple de wrapper TUNe – Langage WDL.

Les interfaces du wrapper définies ici sont : « start », « stop », « configure » et « reconfigure ». Le paramètre « key » définit la classe qui sera exécutée, et « method » la méthode à appeler. Les paramètres d'appels sont aussi décrits. A noter la valeur « $this.srname » qui correspond au nom du composant.

- La spécification du déploiement doit utiliser le langage de description d'architecture de Fractal. TUNe introduit un profil UML (plus intuitif) pour décrire graphiquement le déploiement (voir Figure 2.2.4 et Figure 2.2.5).

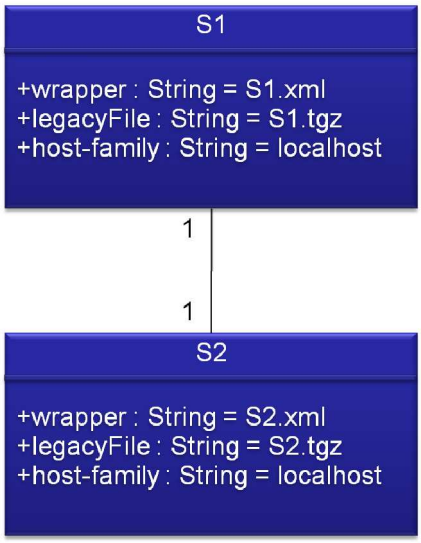


Figure 2.2.4: Exemple de diagramme de déploiement

Les composants S1 et S2 sont décrits ici. Le champ « wrapper » donne le fichier WDL correspondant et le champ « legacyFile » pointe vers le logiciel patrimonial à déployer. Enfin, « host-family » définit la classe utilisée pour la description de l'architecture du parc (voir Figure 2.2.5).

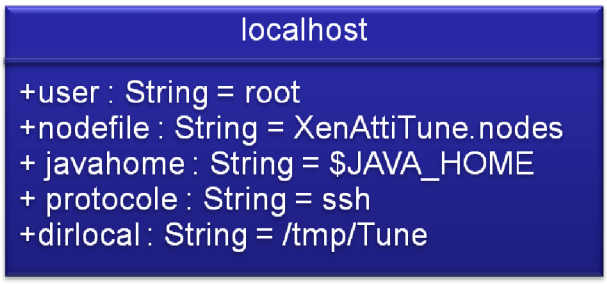


Figure 2.2.5: Exemple de description de l'architecture du parc

Le champ « user » définit l'utilisateur, « nodefile » donne le fichier contenant la liste des noeuds (machines) accessibles par TUNe, « protocole » correspond au protocole de connexion utilisé et enfin « dirlocal » indique le répertoire où TUNe entreprendra ses déploiements.

- Les règles de reconfiguration doivent être écrites en Java et utiliser les API Fractal. TUNe introduit un profil UML pour spécifier les règles de reconfiguration grâce à un diagramme d'états. Ces diagrammes sont utilisés pour définir l'enchainement (*workflow)* des opérations qui doivent être exécutées pour reconfigurer l'environnement administré. Plus précisément, chaque composant TUNe peut communiquer avec l'administrateur TUNe, en envoyant des messages par l'intermédiaire d'un *pipe*. Lors de la réception d'un de ces derniers, l'administrateur parcourt l'ensemble des diagrammes de reconfiguration, et exécute séquentiellement ceux désignés par la requête (voir Figure 2.2.6).

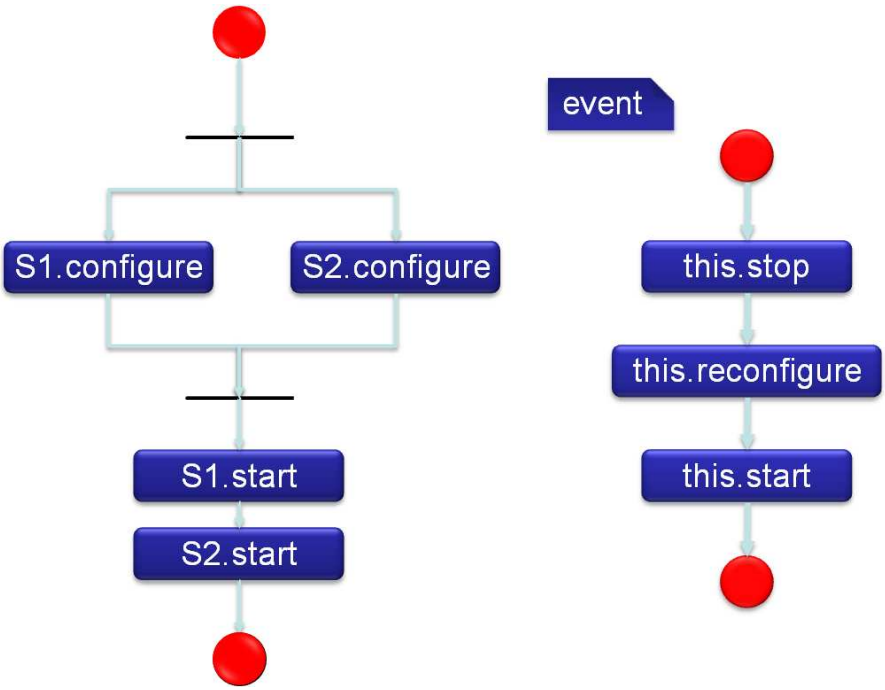


Figure 2.2.6: Exemples de diagrammes de reconfiguration

Le diagramme de gauche présente une succession d'actions, comprenant deux tâches simultanées et un point de rendez-vous.

Le diagramme de droite présente la reconfiguration effectuée par TUNe lors de la réception par l'administrateur de l'évènement « event ».

3 Contribution

3.1 Scénarii implantés

L'implémentation d'un scénario illustratif constitue l'un de nos objectifs. L'idée est simplement de montrer la préservation de la connexion TCP et la faible durée de l'interruption de service. La migration d'un serveur de diffusion vidéo (*streaming*) a été choisie car très explicite.

3.1.1 Scénario illustratif de la migration « à chaud »

Un serveur de streaming est installé sur un domU et diffuse une vidéo.

Le scénario consiste alors en la connexion au serveur de diffusion d'un client indépendant de cette structure. Puis, le serveur est migré et on observe la vidéo pendant tout le temps de la migration. Un résultat concluant est une lecture sans aucune interruption.

La Figure 3.1.1 illustre ce scénario.

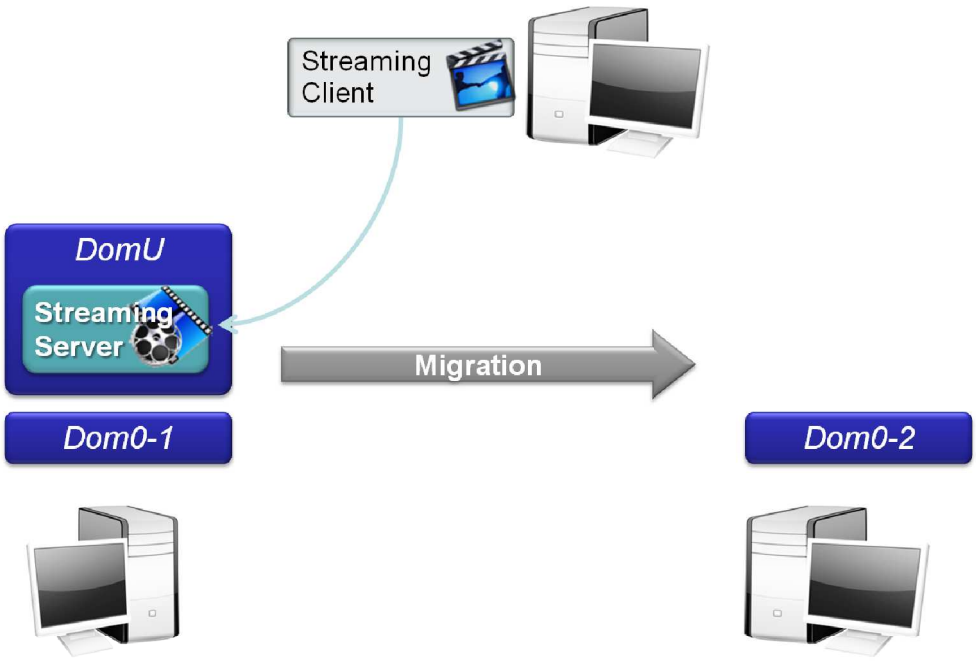


Figure 3.1.1: Scénario illustratif de la migration « à chaud »

3.1.2 Scénario illustratif de la migration autonome

Ce précédent scénario a été déployé et le résultat a été tout à fait concluant. Cependant, il ne démontre pas de capacités d'administration autonome. Un second scénario utilisant TUNe a alors été prévu. Ne pouvant générer un nombre suffisant de connexions au serveur, nous avons décidé de simuler une augmentation de la charge CPU. TUNe est utilisé avec une sonde surveillant la charge CPU et décidant de la migration à partir d'un certain seuil. Le protocole est ensuite le même que

précédemment, la migration n'étant plus manuelle mais pilotée par la sonde.

La Figure 3.1.2 illustre ce scénario.

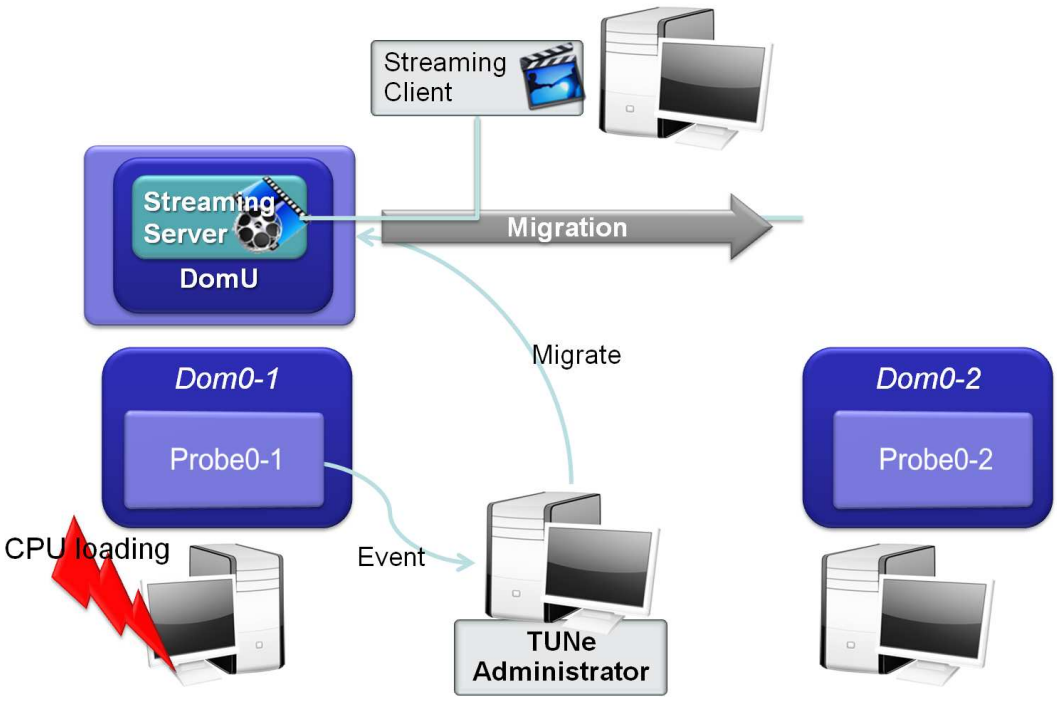


Figure 3.1.2: Scénario illustratif de la migration autonome avec TUNe

3.2 Implantation

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 domaine 0 permettant l'administration des machines virtuelles ainsi qu'un certain nombre de domaines utilisateurs. Leur système de fichiers est situé sur un serveur NFS pour être accessible quelle que soit la machine hôte. Ainsi, les domaines utilisateurs conservent une référence à leur système de fichiers au cours d'une migration.

Il faut distinguer les trois types d'architectures que nous avons mises en place : architecture réseau, architecture logicielle et architecture TUNe.

3.2.1 Architecture réseau

Afin d'élargir notre champ d'action 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 comportant cinq machines dont une exclusivement réservée aux serveurs. En effet, l'isolement de notre réseau local par rapport au réseau mère de l'ENSEEIHT ainsi que certaines contraintes spécifiques au projet nous ont amené à mettre en place quatre serveurs spécialisés :

- un serveur DHCP (Dynamic Host Configuration Protocol)
- un serveur NFS (Network File System)
- un serveur DNS (Domain Name System)
- un serveur NTP (Network Time Protocol)

La totalité du réseau s'articule autour du **serveur DHCP** qui fournit dynamiquement des adresses IP aux machines virtuelles et d'un switch fast ethernet. Dans un premier temps, nous voulions utiliser un hub (élément non actif) mais les performances du hub fourni par le Centre de Compétences en Ressources Informatiques n'étaient pas satisfaisantes.

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 pour répondre à ce besoin. Nous avons opté pour un serveur NFS unique et centralisé, 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 assurée 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.

TUNe désignant les machines par leur nom et non par leur adresse 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.

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

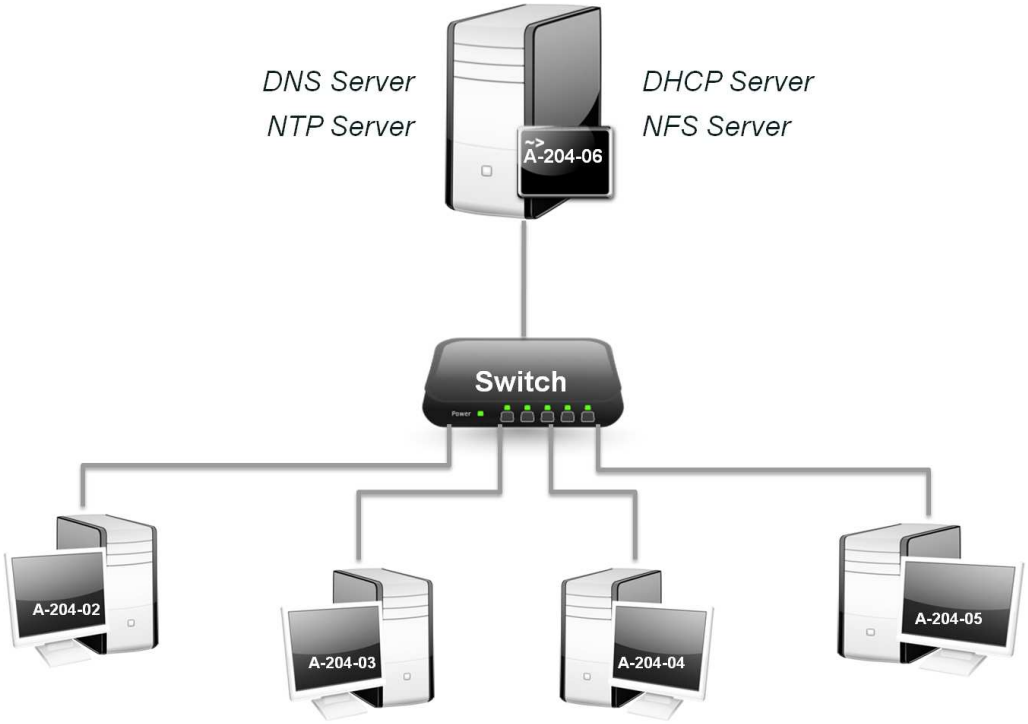


Figure 3.2.1: Architecture réseau

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. De plus, ce service nous offre un moyen de récupération fiable (non sujet aux défaillances locales).

3.2.2 Architecture Logicielle

Il s'agit de la mise en place de Xen (version 3.2) sur les machines non serveurs du réseau. Considérons une machine physique. Elle héberge un dom0 et éventuellement un ou plusieurs domU répondant aux caractéristiques suivantes :

- Le domaine 0 repose sur une distribution Ubuntu 7.10 Gutsy Gibbon pour sa facilité d'utilisation et nos connaissances préalables de son utilisation. Après plusieurs essais, nous avons choisi un noyau Linux version 2.6.18.8 car, d'une part les sources sont disponibles et d'autre part cette version ne présente pas les défaillances observées avec les versions non officielles.
- Les domaines utilisateurs sont construits sur la même version du noyau mais recompilée pour prendre en compte les options nécessaires à un démarrage en NFS (nfs root). Les distributions utilisées sont les suivantes :
  - GNU/Linux Debian Etch (4.02r2) car il s'agit d'une version minimale de Linux.
  - Ubuntu Dapper Drake pour la facilité d'installation et d'utilisation des logiciels dont nous avons eu besoin.

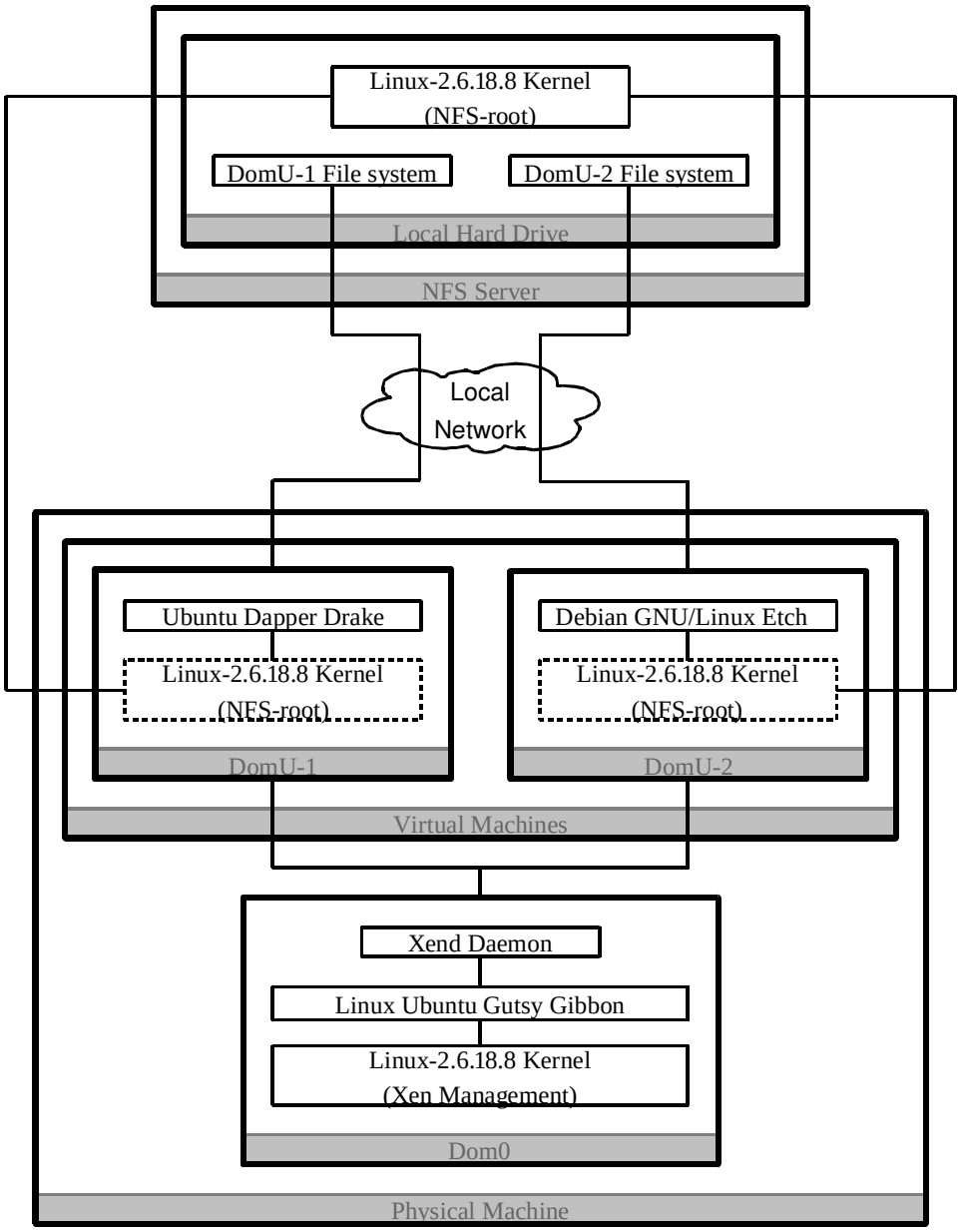


Figure 3.2.2: Architecture logicielle

3.2.3 Architecture 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).

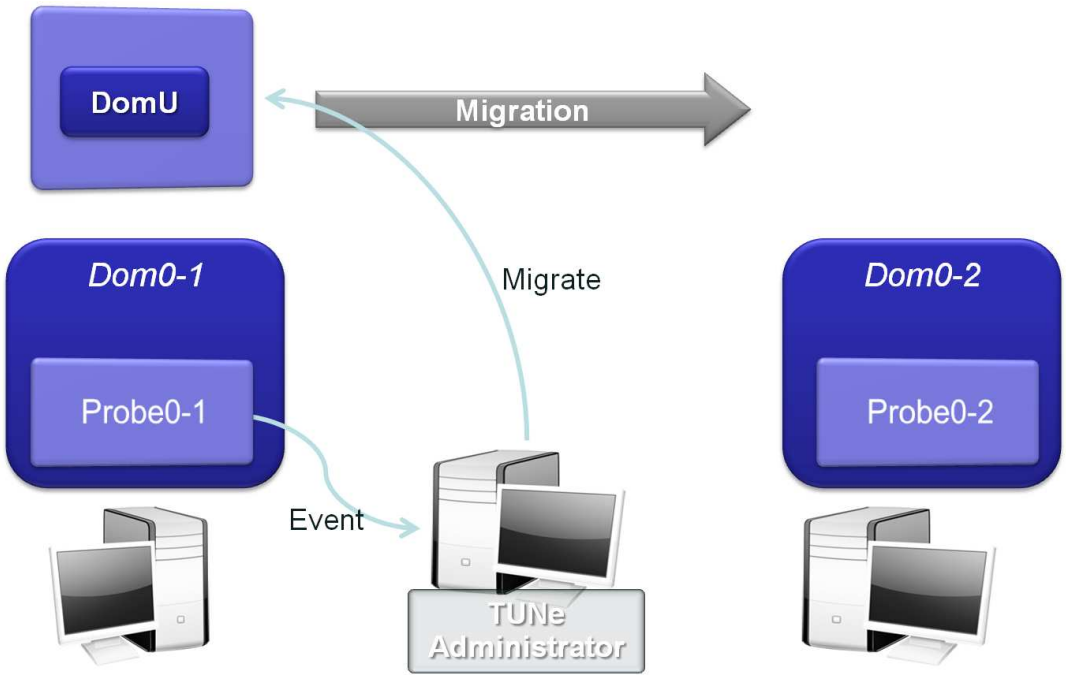


Figure 3.2.3: Architecture globale de l'encapsulation dans TUNe

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.

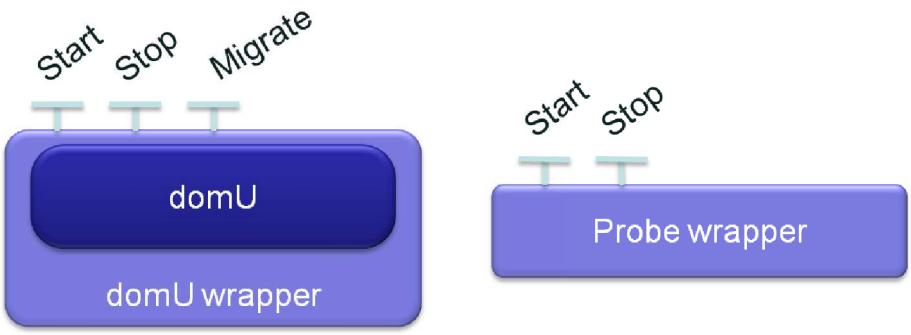


Figure 3.2.4: Wrappers

Le diagramme de déploiement (Figure 3.2.7) 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 3.2.5: 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 3.2.6: Fichier WDL de la sonde développée

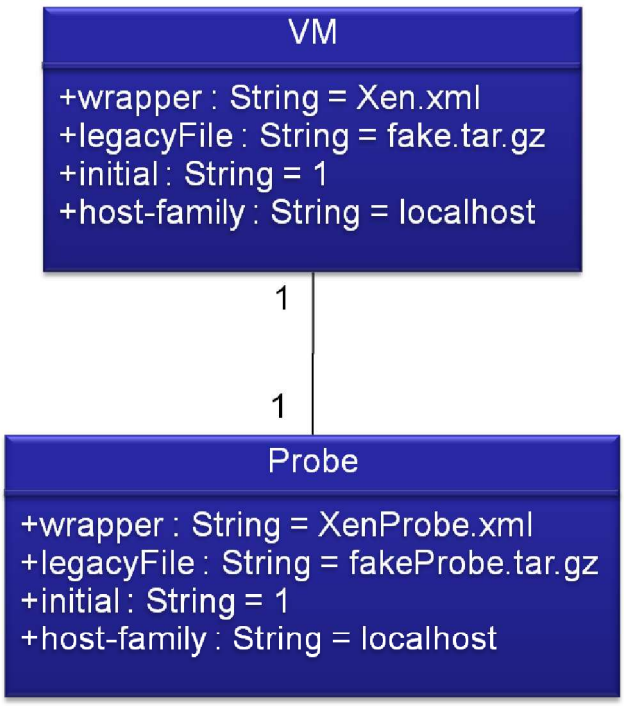


Figure 3.2.7: Diagramme de Déploiement

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.

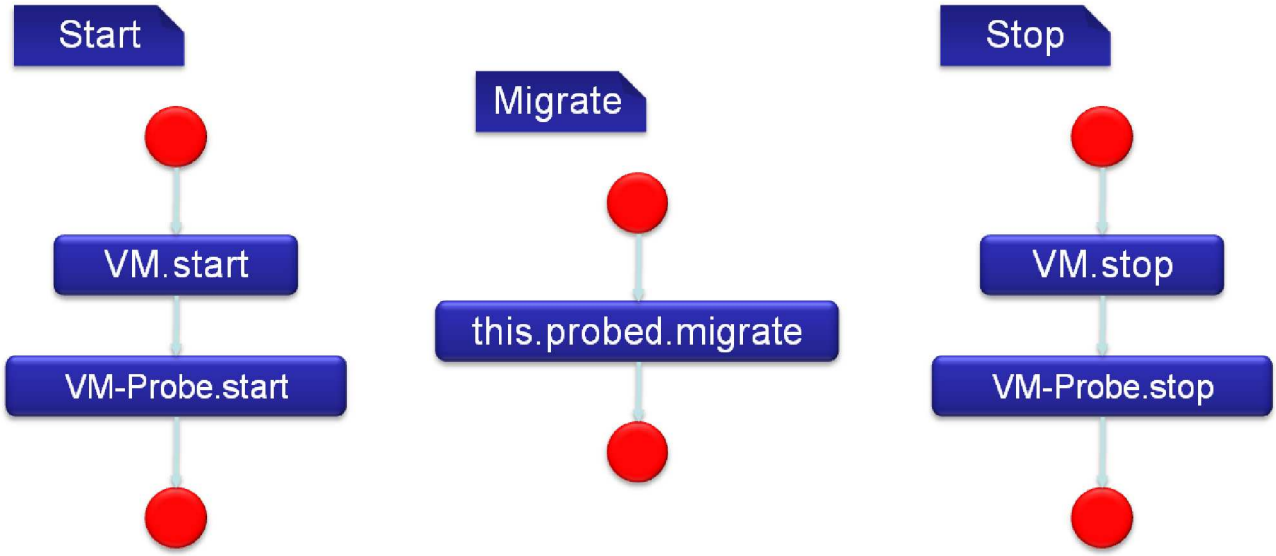


Figure 3.2.8: Diagrammes de Reconfiguration

3.3 Tests et résultats

3.3.1 Considérations préliminaires

Afin de vérifier que les exigences listées dans les spécifications sont bien implémentées dans le produit, des tests de validation sont joués. L'ensemble de tous les tests doit couvrir toutes les exigences et s'inscrire dans le cadre cohérent d'une politique de tests.

Les cas de tests doivent appartenir à des classes d'équivalence indépendantes représentatives du plus grand éventail possible de situations concrètes. La campagne de tests doit se dérouler de manière itérative. En effet, si tous les tests sont déroulés, alors tous les dysfonctionnement apparaissent et peuvent ensuite être traités. Ce n'est que lors d'une seconde itération que les tests sont rejoués après correction de l'application.

Dans le cadre du projet Xen AttiTUNe, le développement est divisé en deux phases distinctes. La première consiste à mettre en place des scénarii utilisant la migration de machines virtuelles dans un environnement distribué. La seconde consiste à intégrer ces scénarii à TUNe dans le but d'automatiser la migration.

3.3.2 Plan de tests

Le plan de tests se scinde en trois parties complémentaires. La première regroupe les tests tendant à vérifier les propriétés du mécanisme de migration. La deuxième est constituée des tests ayant pour but l'étude des performances du mécanisme de migration. La troisième doit valider la stabilité des propriétés du mécanisme de migration lors de son intégration à l'environnement TUNe.

Certains tests, notamment ceux concernant la mesure de performances, amènent à analyser les caractéristiques propres à la migration, e.g. sa durée ou le délai qu'elle introduit dans le temps d'exécution d'un programme. Les mesures de performances suivent un protocole expérimental qui s'organise autour de deux cas, le test témoin, qui fait appel à des outils déjà validés de l'environnement de développement et le test spécifique au projet.

Le plan de test se présente sous la forme d'une séquence de tests. À chaque test est associée une référence à l'exigence validée permettant d'assurer la traçabilité des exigences tout au long du projet. Les prérequis du test, son déroulement et les résultats attendus sont explicités. Le plan de test est conçu parallèlement à la phase de spécification.

3.3.3 Rapport de tests

3.3.3.1 Considérations générales

La campagne de tests est basée sur les scénarii présentés dans le plan de tests. Afin d'assurer la traçabilité des exigences validées, chaque test fait l'objet d'un rapport qui décrit son déroulement et les conclusions qui peuvent en être tirées. De plus, la technologie mise en œuvre lors du test profitera au rapport sous la forme d'une quantification des performances.

Les programmes de tests seront intégrés au rapport de manière à offrir une méthode standard de validation. Les résultats de ces programmes seront analysés automatiquement afin d'assurer l'objectivité de l'adéquation entre résultats obtenus et attendus.

3.3.3.2 Problèmes rencontrés

Dans le cas du projet Xen AttiTUNe, la logique suivie pour la résolution de problèmes repose sur la documentation en ligne et l'appel à des spécialistes techniques appartenant au cadre du projet.

Une fois le problème identifié et reproduit, tout le travail effectué jusqu'au point courant doit être sauvegardé sous la forme de documentation, notamment de tutoriels, et de rapports de tests. La *baseline, i.e.* la liste des versions des programmes utilisés pour effectuer les tests, doit être tracée pour identifier l'état de l'application complète lors du test.

Ensuite, la phase de résolution proprement dite consiste à mettre à jour les programmes à la lumière des dysfonctionnements observés. Pour ce faire, des solutions techniques sont explorées en parallèle et évaluées en fonction de leur compatibilité avec l'état courant du système. Ceci nécessite une organisation basée sur la liste d'actions qui doit donc être quotidiennement mise à jour.

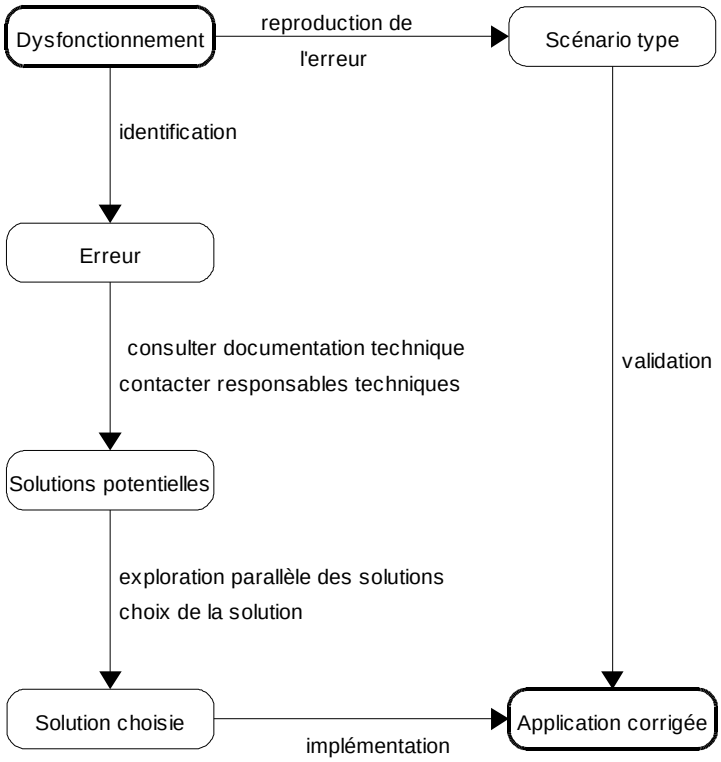


Figure 3.3.1: Traitement des problèmes techniques.

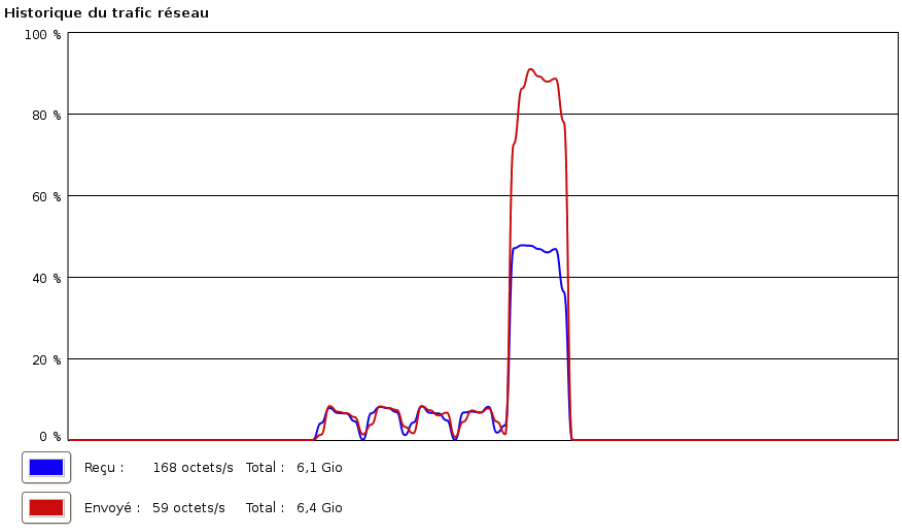
Lors de la campagne, certains dysfonctionnements de l'architecture ont été mis à jour. La désynchronisation des domUs lors de la migration d'une machine physique à l'autre en est l'exemple le plus représentatif. En effet, la résolution a demandé un remaniement en profondeur de l'architecture afin d'aligner les versions des noyaux des dom0s et domUs pour assurer la synchronisation au sein de chaque machine physique. D'autre part et parallèlement à la première tâche, un serveur NTP a été installé pour garantir la synchronisation entre les dom0s.

3.3.3.3 Mesure des performances

La mesure des performances relatives à la migration est un des objectifs du projet Xen AttiTUNe. Aussi les conclusions concernant la mesure de la durée de migration et de l'interruption qu'elle engendre seront présentées ici. C'est aussi l'occasion de prendre connaissance de certains résultats expérimentaux obtenus lors des tests.

Le scénario suivant permet de constater les effets de la migration d'un domU écrivant dans un fichier sur l'activité du réseau au niveau des machines extrémités.

[1] Activité réseau sur dom0_a-204-02



[2] Activité réseau sur dom0_a-204-03

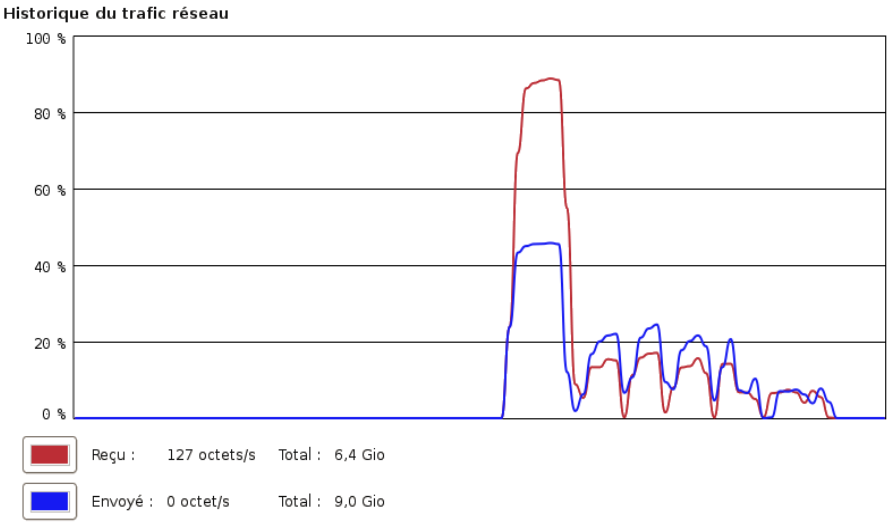


Figure 3.3.2: Activité réseau liée à l'écriture dans un fichier situé sur un serveur NFS.

Le système de fichier du domU est hébergé par un serveur NFS.

[1] Sur dom0_a-204-02, qui est le dom0 source, avant la migration, la communication en créneau enregistrée correspond à l'écriture dans le fichier situé sur le serveur NFS distant.

[2] Sur dom0_a-204-03, le dom0 destination, la même forme d'activité est enregistrée sur le réseau et elle disparaît totalement sur dom0_a-204-03. Ceci s'explique par le fait que le programme d'écriture dans le fichier s'exécute sur le domU qui a subi une migration de dom0_a-204-02 vers dom0_a-204-03. L'activité sous forme de créneau provient du fait que les données à écrire sont d'abord enregistrées dans un buffer, dans la mémoire locale contenue dans le domU et donc gérée au niveau du dom0 correspondant, puis envoyées pour mettre à jour le fichier sur le serveur NFS.

La Figure 3.3.3 décrit l'activité réseau liée à deux migrations successives d'un domU hébergeant un serveur de streaming vidéo.

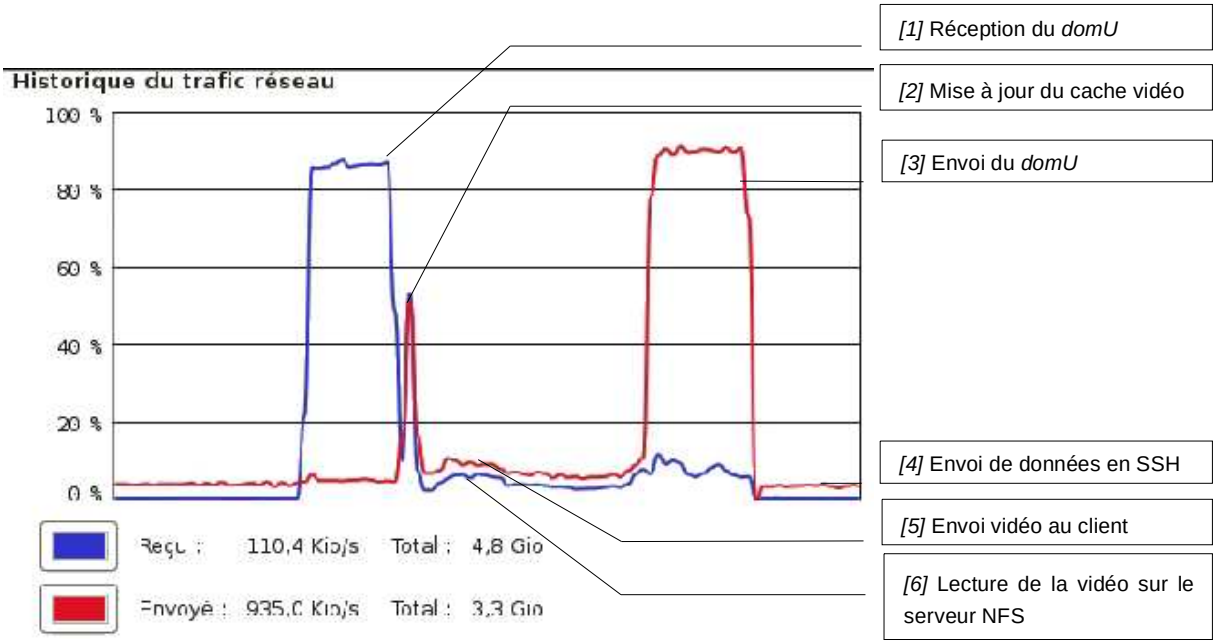


Figure 3.3.3: Activité réseau pendant la migration d'un serveur de streaming.

Le domU héberge un serveur de streaming vidéo dont dom0_a-204-05 est un client. Le scénario de test consiste à opérer une migration du domU de dom0_a-204-02 vers dom0_a-204-03.

[1] Le débit réseau le plus élevé est dû à la réception du domU et [3] à son envoi.

[2] Le pic enregistré après la réception du domU s'explique par la lecture sur le serveur NFS du cache sauvé par le client, alors que le domU se trouvait encore sur dom0_a-204-02, et à la restitution de ce cache au client.

[3] L'envoi du domU vers dom0_a-204-02 est conjugué à l'envoi de la vidéo vers le client, à savoir dom0_a-204-05. De même, la communication avec le serveur NFS est à considérer comme la somme des données vidéo reçues dans le but d'une transmission au client.

[4] Une connexion SSH permet de suivre l'évolution de l'activité réseau sur dom0_a-204-03. C'est pourquoi le flux de données émis ne passe jamais sous un certain seuil correspondant au flux de la connexion SSH.

[5] Le système de fichiers du domU est situé sur le NFS, donc il lui correspond un accès réseau lors de la réception du domU par dom0_a-204-03 puis [6] l'envoi des données lues au client.

Après avoir introduit le principe de migration et ses effets de bord sur le trafic réseau, il s'agit désormais de mesurer la durée de la migration ainsi que l'interruption qu'elle introduit. C'est le but de la Figure 3.3.4.

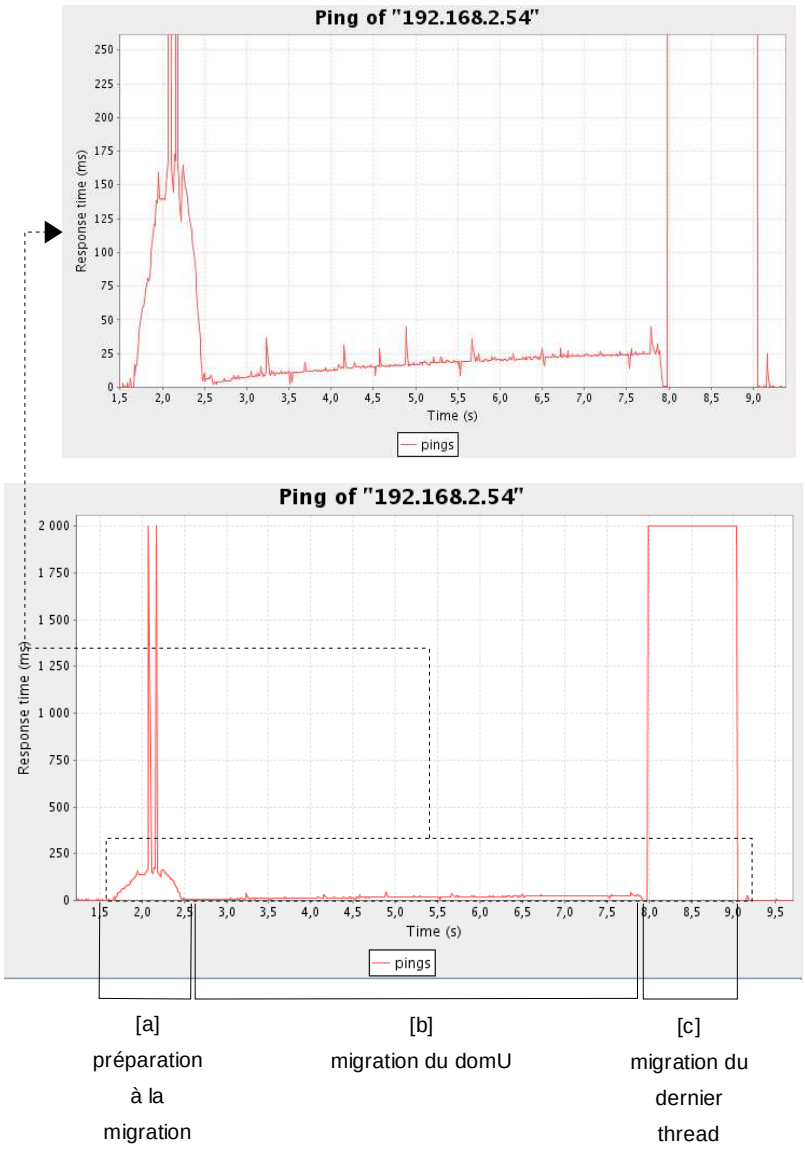


Figure 3.3.4: Mesure de l'interruption introduite par la migration.

Le graphique présente le temps de réponse à un ping (dont la fréquence est de 1 ping toutes les 10 ms le timeout vaut 2 s) d'un domU en cours de migration à partir d'une machine physique distante. En dehors de la migration, le temps de réponse est en général inférieur à 10 ms.

[a] La première phase de la migration consiste à vérifier la connexion avec le dom0 distant vers lequel ont veut migrer. La présence de deux timeout lors de cette phase peut s'expliquer par l'exécution de deux opérations atomiques de longue durée.

[b] Lors de la migration du domU, le temps de réponse au ping augmente sensiblement. Ceci s'explique par l'augmentation du nombre de tâches intrinsèques à la migration à effectuer au niveau du domU pour assurer le transfert complet du domaine.

[c] La troisième phase correspond à la migration du dernier thread qui s'exécute et qui s'arrêtera pour 1 s durée de l'interruption du domU figurée ici par une colonne de pings atteignant chacun leurs timeout. La totalité de la migration dure ici environ 8 s, ce qui correspond à l'ordre de grandeur des temps de migration observés..

3.4 Conclusion

La couverture des exigences par les tests est résumée dans la matrice de couverture (Figure 3.4.1). Les symbole de succès ou d'échec sont situés à l'intersection de la ligne du test considéré et de la colonne de l'exigence qu'il permet de valider. Les tests et exigences sont détaillés dans les documents fournis en annexe, en particulier dans les *Spécifications* et le *Plan de test*.

		Exigences											
		EF1	EF2	EF3	EF4	EF5	EF6	EF7	EF8	EF9	EF10	EF11	EF12
Tests	T1	●											
	T2	●	●										
	T3	●		●									
	T4	●	●	●	●								
	T5	●				●							
	T6	●					●						
	T7	●						●					
	T8	●	●						●				
	T9	●		●						●			
	T10	●			●						●		
	T11	●										●	
	T12	×											×

Figure 3.4.1: Matrice de couverture.

Le symbole ● indique que l'exigence est vérifiée par le test considéré.
Le symbole × indique que le test correspondant n'a pas été mis en œuvre.

Table des Figures

4 Annexe

4.1 Spécifications

4.2 Architecture

4.3 Plan de tests

4.4 Rapport de tests

4.5 Tutoriels

4.5.1 Tutoriel de mise en place de l'architecture globale

4.5.2 Installation de Xen 3.2.0 avec noyau linux 2.6.18.8

4.5.3 Guide d’installation du DHCP

4.5.4 Tutoriel d'installation de NFS

4.5.5 Tutoriel d'installation d'un serveur DNS : BIND

4.5.6 Tutoriel d'installation d'un serveur NTP : ntp-server

4.5.7 Tutoriel d'utilisation de TUNe