Institut **S**upérieur
d'**I**nformatique, de
Modélisation et de
leurs **A**pplications

BP 10125
6173 Aubière Cedex

Hutchinson SA

Centre de Recherche
BP 31, rue Gustave Nourry
45120 Chèlette-sur-Loing

Rapport d'ingénieur
Stage de 2nd année
Filière : Génie Logiciel et Systèmes Informatiques

Virtualisation d'outils scientifiques

Présenté par : Julien PINGUET
Sous la direction de : Yann COLLETTE

Durée : 5 mois
Avril-août 2012

Remerciements

Avant toute chose, je tiens à adresser mes remerciements aux personnes qui ont fait que mon stage se passe dans les meilleurs conditions.

Je remercie tout d'abord Jean-Luc Sortais, chef du service de simulation numérique, pour m'avoir accepté en tant que stagiaire.

J'adresse aussi mes remerciements à Yann Collette, pour son encadrement durant ces 5 mois, son aide lorsque je rencontrais des difficultés techniques malgré sa charge de travail, ainsi que pour le degré de liberté qu'il a pu m'accorder.

Je remercie Sophie Genieys qui m'a aidé lors des sollicitations techniques ou méthodiques.

Enfin, je remercie David Charette, étudiant à l'ISIMA qui a effectué son stage dans la même entreprise, et qui a su me conseiller du mieux qu'il pouvait.

Résumé

Mon stage de seconde année d'étude à l'ISIMA avait pour but de développer une solution de virtualisation, permettant d'exécuter des outils scientifiques uniquement disponible sous Linux.

Tout d'abord j'ai construit et configuré une première machine virtuelle Linux complète, sur laquelle se basera ma première solution. Son environnement de développement complet permettra de compiler des outils scientifiques.

Grâce à l'analyse des différentes techniques existantes et utilisables, j'ai pu m'orienter vers les solutions que j'utiliserai. Il fallait aussi penser à obtenir la solution la plus pratique pour l'utilisateur mais aussi pour son déploiement dans l'entreprise.

Une fois les programmes développés et les tests effectués, il ne restait plus qu'à améliorer les différents points qui posent problème avant de construire une machine virtuelle minimale qui offrira de meilleures performances.

Mots clé : Virtualisation, VirtualBox, VBoxManage, Outil scientifique, Python, wxPython

Abstract

During my ISIMA second year studies, my work placement's aim was to build a virtualization solution in order to run scientific tools only available under Linux.

I have first built and configured a full Linux virtual machine, on which my first solution will be based. Its full development environment will compile scientific tools.

In analyzing the different techniques that could exist and be usable, I could turn towards the solutions I will use. I have to think about the most practical solution for users but also for its development in the firm.

It will be necessary to improve the different remaining problems once the programs are developed and tests are performed. Then, a new minimal virtual machine that will offer better performance could be built.

Keywords : Virtualisation, VirtualBox, VBoxManage, Scientific tool, Python, wxPython

Table des matières

Table des figures

Introduction

En informatique, la compatibilité est la capacité d'un logiciel ou d'un matériel de fonctionner sur les différents systèmes, sans en altérer le fonctionnement. Dans le domaine du logiciel, cela peut concerner un système d'exploitation, un programme, ou même une librairie. Le problème de compatibilité le plus connu est celui entre Windows, marque propriétaire, et Linux, solution gratuite et ouverte à tous.

Dans le domaine de la recherche par exemple, il existe des outils scientifiques qui ne sont compatibles uniquement sous Linux. Lorsque les utilisateurs possèdent un ordinateur ayant comme système d'exploitation Windows, il leur est donc impossible de les utiliser de manière simple.

Cependant, il existe plusieurs techniques permettant de contourner ce problème, chacune présentant des avantages et des inconvénients.

- La cross-compilation, qui consiste à compiler le programme sur la machine compatible avant d'exporter le programme vers la machine désirée, n'est possible que dans de rares cas.
- L'installation d'un autre système d'exploitation, compatible avec l'outil, sur une autre partition du disque-dur, appelé le "multi-boot", demande un redémarrage de l'ordinateur entre chaque échange de système d'exploitation.
- Enfin, la virtualisation, quant à elle, consiste à émuler un système d'exploitation à l'intérieur même du système d'exploitation utilisé. Cette solution est possible dans tous les cas, et permet le meilleur compromis entre temps de mise en œuvre, et puissance de calcul.

L'objet de cette étude sera d'étudier et de développer une solution de virtualisation, simple d'utilisation voir transparente pour l'utilisateur, qui permettra d'utiliser des outils à partir d'une interface graphique.

La construction d'une première machine virtuelle complète permettra de tester et de comparer les différentes techniques possibles et utilisables dans la solution finale. Une fois ces techniques

définies, elles seront optimisées et intégrées à une seconde machine virtuelle minimale pour obtenir les meilleurs résultats possibles. Enfin, l'intégration de la solution dans une interface graphique permettra l'utilisation pratique et efficace d'outils, à l'origine non exécutables.

Tout d'abord, j'analyserai précisément le sujet, en présentation l'entreprise, la virtualisation, ainsi que la solution envisagée. Dans un deuxième temps je développerai la démarche suivie pour parvenir à la solution finale du projet. Pour terminer, je présenterai le développement et la solution obtenue.

I Introduction de l'étude

I.1 Présentation de l'entreprise

I.1.1 L'entreprise

Le groupe Hutchinson est une entreprise spécialisée dans la transformation du caoutchouc industriel, dont l'innovation est l'objectif principal. Plusieurs domaines de compétences et technologiques sont mis en œuvre par l'entreprise :

- l'étanchéité,
- l'isolation (thermique, acoustique, ou vibratoire),
- le transfert de fluides,
- la transmission et la mobilité,
- la protection et les soins.

Hutchinson est implanté dans 20 pays dans le monde, avec plus de 29.000 collaborateurs répartis dans 91 sites.

En 2011, son chiffre d'affaires s'élève à 2.987 millions d'euros, dont 64% provient du secteur de la transmission et mobilité, et 63% en Europe.

Créée en 1853, l'entreprise devient une filiale de Total, 5ème groupe pétrolier mondial, suite à la prise majoritaire des parts du groupe Hutchinson.

I.1.2 Recherche et développement (R&D)

La R&D (Recherche et Développement) est un point fort du groupe Hutchinson. En effet, cela permet à l'entreprise de concevoir des produits à la pointe de la technologie, et de rester le plus compétitif. 30 brevets ont ainsi été déposés en 2010.

Le groupe Hutchinson possède un centre de recherche, basé à Montargis à proximité de Paris (France), ainsi que 26 centres de développement techniques dans le monde.

Au total, ce sont près de 2.000 ingénieurs et techniciens qui travaillent dans la R&D, majoritairement dans le domaine de la chimie, la physique et la mécanique. L'entreprise y consacre 144 millions d'euros en 2011, soit 4,8% de son chiffre d'affaire, mais aussi 15% de ses ressources.

I.1.3 Lieu du stage

Le développement de logiciel de simulation est un des secteurs d'activité de la R&D.

Le stage s'est effectué dans le centre de simulation numérique, situé à Levallois-Perret (France). Le service est composé de 2 ingénieurs, et emploie plusieurs intervenants extérieurs, qui sont chargés de développer et maintenir les nombreux logiciels utilisés par les différents secteurs de l'entreprise :

- les logiciels métier, utilisés par les industriels,
- les logiciels de simulation numérique, utilisés principalement dans le centre de recherche.

I.2 Qu'est ce que la virtualisation ?

Avant de commencer ce rapport, quelques explications sur la virtualisation permettront de mieux comprendre cette solution, sur laquelle se base mon projet.

I.2.1 Objectif

La virtualisation consiste à émuler un ou plusieurs systèmes d'exploitation sur un même ordinateur.

Il est ainsi possible d'utiliser par exemple Linux au sein même d'un environnement Windows.

On appelle "machine hôte" (ou "l'hôte") le système principal installé sur l'ordinateur, et on appelle "machine virtuelle" le système d'exploitation qui sera virtualisé et qui fonctionnement au sein de la machine hôte.

Dans ce projet, Windows sera la machine hôte et Linux sera la machine virtuelle.

La capture d'écran I.1 illustre simplement le principe de la virtualisation. Il s'agit du système d'exploitation Linux (Fedora 16) fonctionnant à l'intérieur d'une simple fenêtre Windows.

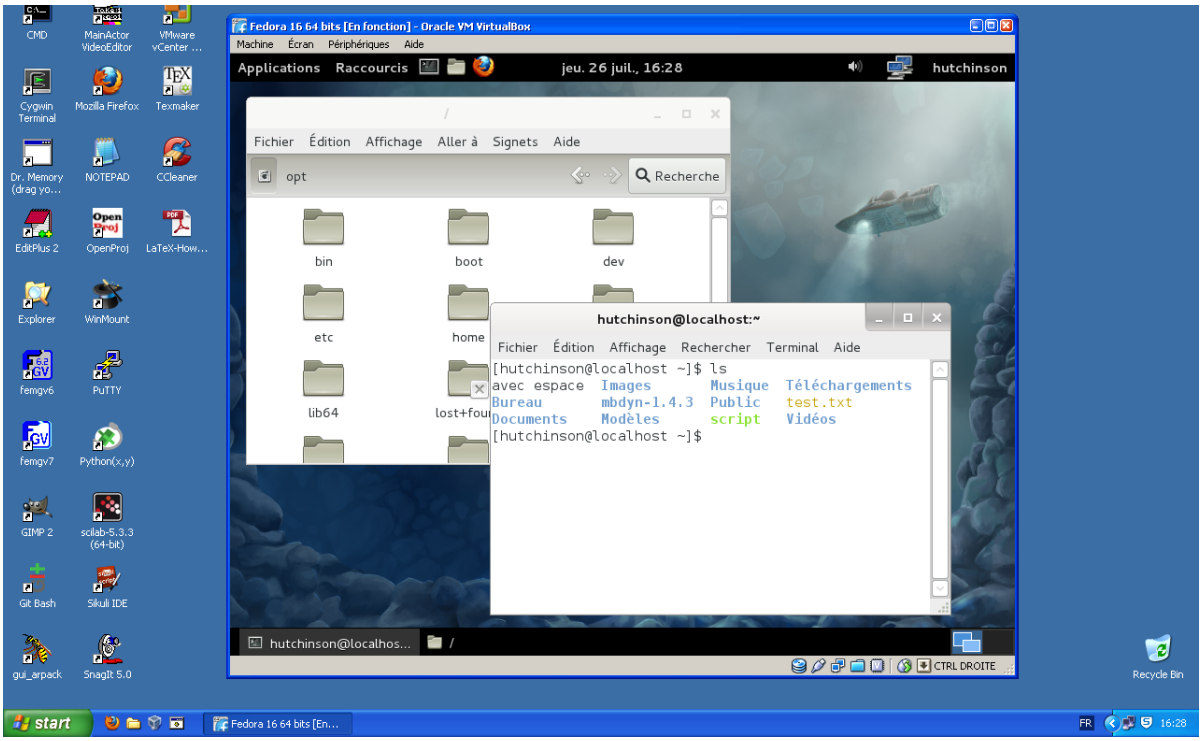

FIGURE I.1 – Système d'exploitation Linux virtualisé dans un environnement hôte Windows

I.2.2 Fonctionnement

Le principe général de la virtualisation consiste à créer une interface, ou couche d'abstraction, entre le matériel de l'ordinateur et le système virtualisé. Cette couche est appelée *hyperviseur*.

Il existe plusieurs techniques différentes permettant la virtualisation. Ce qui les différencie est le degré d'abstraction et l'agencement des couches. Les deux types de virtualisations les plus utilisés vous sont présentés ici.

I.2.2.1 Hyperviseur de type 1

Le premier type de virtualisation est appelé *hyperviseur de type 1*. L'hyperviseur permet de gérer l'accès des noyaux invités au matériel, comme présenté dans la figure I.2.

Cette technique est la plus performante bien que coûteuse et compliquée à mettre en place. Elle est principalement utilisée sur des serveurs qui disposent de plusieurs fonctions regroupées sur une même machine par gain d'énergie ou d'investissement.

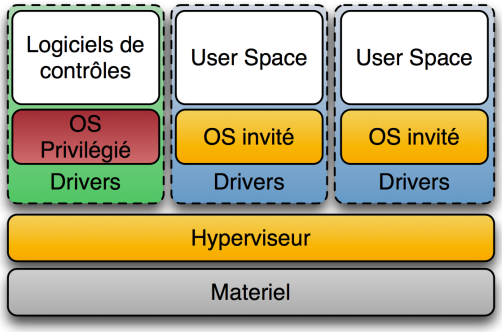

FIGURE I.2 – Hyperviseur de type 1
Source : Wikipédia - http ://fr.wikipedia.org/wiki/Virtualisation

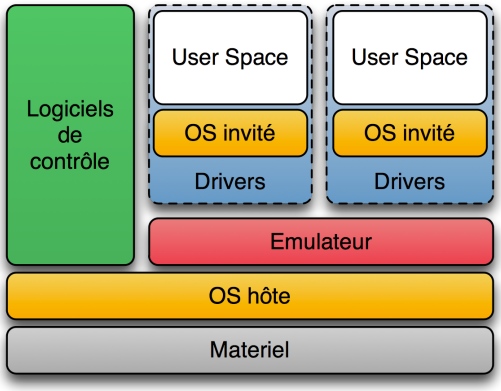

FIGURE I.3 – Hyperviseur de type 2
Source : Wikipédia - http ://fr.wikipedia.org/wiki/Virtualisation

I.2.2.2 Hyperviseur de type 2

Dans cette technique, l'hyperviseur est un logiciel fonctionnant sur le système d'exploitation hôte (voir figure I.3). Il a pour but de virtualiser ou d'émuler les composants de l'ordinateur (processeur, mémoire vive, mais aussi carte graphique, carte son, carte réseau, ...). Ainsi, les machines virtuelles fonctionneront comme si elles communiquaient avec de vrais composants.

Bien que les performances des machines virtuelles soient inférieures au type 1, cette solution est très simple à mettre en place car elle nécessite principalement l'installation du logiciel de virtualisation.

Dans ce projet, il s'agira d'utiliser cette technique de virtualisation pour réaliser la solution. Nous nous restreindrons donc à l'hyperviseur de type 2 dans la suite de ce rapport.

I.2.3 Particularités de la virtualisation

Plusieurs particularités seront mis en évidence et développées tout au long de ce rapport.

I.2.3.1 Les composants virtualisés

Dans le cas d'une machine physique, la modification des performances demande un changement des composants, tels que le processeur, les barrettes de RAM, la carte graphique, ...

L'avantage de la virtualisation, dans le cas de l'hyperviseur de type 2, repose sur le fait que les composants matériels sont virtualisés ou émulés par l'hyperviseur. Ainsi, il est possible de régler facilement les performances de la machine virtuelle en modifiant les ressources allouées. Cela se fait généralement en utilisant les outils de configuration mis à disposition par le logiciel de virtualisation.

Selon le logiciel utilisé, les composants seront virtualisés ou émulés. La virtualisation possède toutefois des performances plus élevées que l'émulation.

I.2.3.2 Les disques virtuels

On appelle *disque virtuel* la mémoire allouée permettant d'émuler un disque dur.

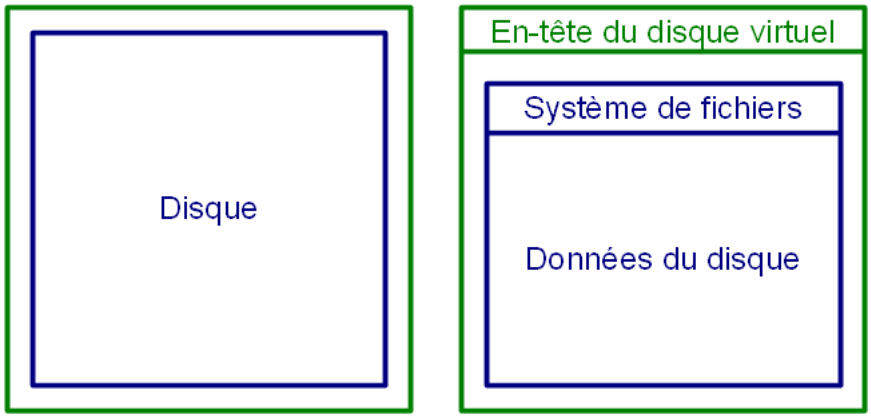

FIGURE I.4 – Schéma de la structure d'un disque virtuel

Le disque virtuel sera modélisé sous forme d'un fichier (d'extension *.vdi*) présent dans l'arborescence du système d'exploitation hôte. Ce fichier encapsule un disque, et est composé de plusieurs parties :

- un en-tête, contenant plusieurs informations sur la structure du fichier ;
- la structure d'un disque composé du système de fichiers et des données associées.

La figure I.4 schématise la structure globale du fichier.

Comme pour les machines physiques où l'on peut brancher des disques durs sur des ports SATA [1] de la carte mère, il est possible d'ajouter des disques virtuels à la machine virtuelle pour émuler l'ajout d'un disque dur. Cette action est effectuée par l'hyperviseur, et la machine virtuelle le considérera comme un vrai disque.

I.2.3.3 Les types de réseau

La carte réseau est un des composant nécessaire à la machine virtuelle pour accéder à la machine hôte ou au réseau extérieur. Le composant est virtualisé (voir partie I.2.3.1), ce qui permet de définir quel type de réseau sera établi entre la machine hôte et la machine virtuelle.

Accès privé hôte : Chacune des machines virtuelles a accès à la machine hôte, mais n'a aucun accès à l'extérieur du réseau (schéma I.5). Ce type de réseau ne permet pas d'accéder à internet, et les machines virtuelles ne sont pas joignables depuis l'extérieur.

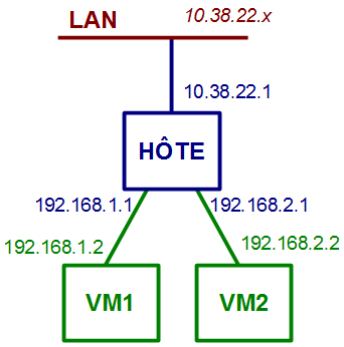

FIGURE I.5 – Réseau de type "Accès privé hôte"

Réseau interne : Ce mode permet de créer un nouveau réseau privé entre les différentes machines virtuelles de la machine hôte (schéma I.6). Les machines virtuelles n'auront accès qu'à ce réseau, et ne pourront pas accéder au LAN ni à internet.

1. SATA (Serial Advanced Technology Attachment) : Norme utilisée pour l'échange de données entre les cartes mères et les stockages de masse. Donne son nom aux ports utilisés pour le branchement des disques car un "câble SATA" est nécessaire.

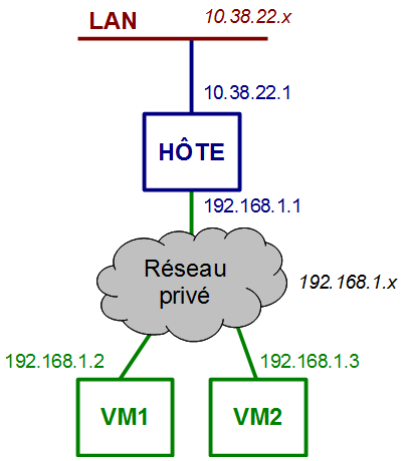

FIGURE I.6 – Réseau de type "Réseau interne"

NAT : Les trames de la machine virtuelle transiteront par le biais de la machine hôte, qui servira de routeur. Une translation d'adresse (NAT, Network Address Translation) sera effectuée, et les paquets possèderont la même adresse IP que la machine hôte. (schéma I.7)

La machine virtuelle aura accès au réseau LAN, mais vu du LAN la machine virtuelle n'existera pas. Elle ne pourra pas être utilisée comme serveur, mais uniquement comme client.

Ce type de réseau est le plus pratique pour accéder à internet et pour une utilisation basique, sur un réseau sans configuration spéciale (proxy, IP fixe, ...).

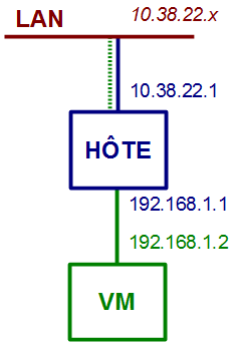

FIGURE I.7 – Réseau de type "NAT"

Accès par pont (bridge) : Ce mode est le plus complet, car il permet à la machine virtuelle d'apparaitre sur le réseau comme s'il s'agissait d'une nouvelle machine (schéma I.8). Elle possèdera ainsi sa propre adresse IP et adresse MAC, et sera accessible par les machines du LAN. Il est ainsi possible d'utiliser la machine virtuelle comme serveur.

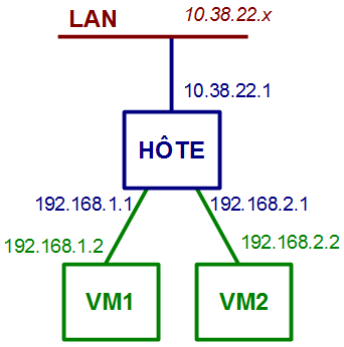

FIGURE I.8 – Réseau de type "Accès par pont"

I.2.3.4 Le répertoire partagé

Il est possible de partager un répertoire, appelé *répertoire partagé*, entre la machine hôte et la machine virtuelle. Le contenu sera ainsi disponible "simultanément" sur la machine hôte et la machine virtuelle.

Le principe est le même que le "répertoire partagé" du réseau Windows, où les utilisateurs du réseau pourront accéder au répertoire, à l'exception qu'il ne demande aucune configuration particulière (droits d'accès, réseau, ...).

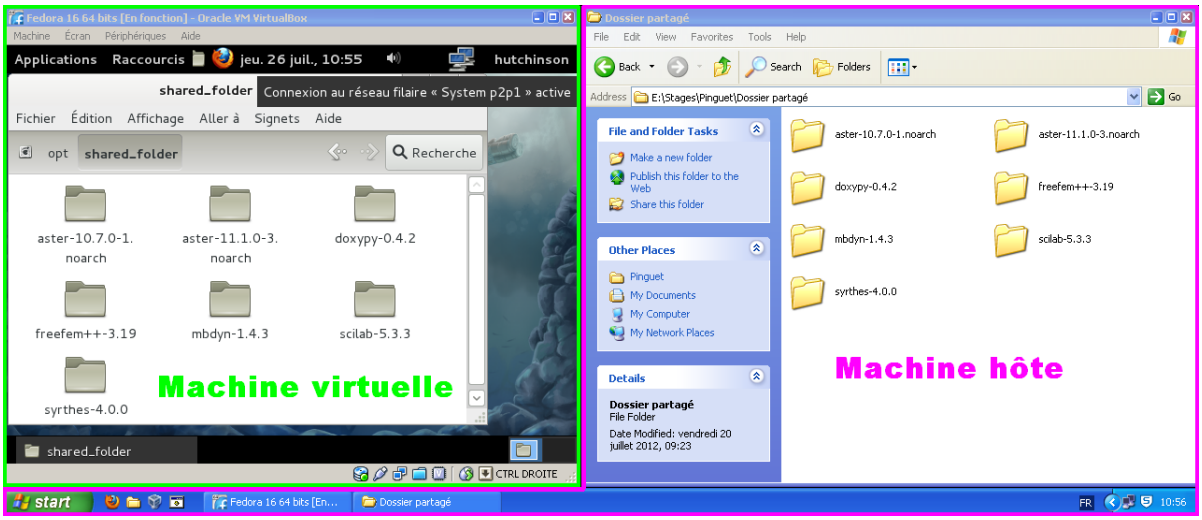

FIGURE I.9 – Répertoire partagé

La capture d'écran de la figure I.9 illustre le principe du répertoire partagé. Dans cet exemple, la machine hôte (Windows) va partager le répertoire `E:\Stage\Pinguet\Dossier partage\`

avec la machine virtuelle, qui va le monter dans le répertoire `/opt/shared\_folder`.

Cela requiert l'installation des *additions invitées* sur la machine virtuelle, aussi appelées *addons*, qui fournissent de nombreux avantages pour les utilisateurs en améliorant la virtualisation :

- un affichage graphique de meilleure qualité, dont la 2D et 3D améliorées, l'adaptation de la résolution graphique à celle de la fenêtre ;
- le partage du presse-papier ;
- la capture ou la libération du pointeur de la souris lorsque l'on entre ou sort de la fenêtre de la machine virtuelle ;
- l'utilisation du répertoire partagé.

I.3 Étude du problème

I.3.1 Besoin pour les utilisateurs

Dans l'entreprise Hutchinson, tout particulièrement dans le centre de recherche, des outils scientifiques utilisés ne sont compatibles que sur les systèmes d'exploitation Linux. C'est le cas par exemple de Code Aster. Or, le système d'exploitation déployé au sein de l'entreprise est Windows, ce qui ne permet pas d'utiliser ces outils.

I.3.2 Première solution

La première solution qui a été envisagée par la section informatique du centre de recherche, a été de déployer une machine virtuelle, de type VirtualBox ou VMware, sur l'ensemble des ordinateurs.

Cette solution est fonctionnelle, mais possède deux inconvénients non-négligeables :

- Tout d'abord, l'ensemble de la machine virtuelle était installée sur un seul et même disque virtuel, ce qui pose un gros problèmes lors du déploiement de la machine virtuelle. En effet, le système d'exploitation (complet) seul peut atteindre la taille d'une dizaine de GigaOctets, et chacun des outils plusieurs centaines de MégaOctets. De plus, lors d'une mise à jour d'un simple outil, il est nécessaire de redéployer l'ensemble de la solution, c'est à dire le disque virtuel.
- Le second inconvénient repose sur l'utilisation de la machine virtuelle. Les utilisateurs doivent actuellement configurer et faire fonctionner la machine virtuelle, avant de pouvoir exécuter les outils dont ils ont besoin. Le programme de virtualisation, comme par exemple VirtualBox, ou même le système d'exploitation Linux utilisé, n'est pas forcément facile à

utiliser, et demande tout de même certaines compétences en informatique.

I.3.3 Solution envisagée

En analysant la solution actuelle, tout particulièrement les deux problèmes principaux, notre objectif était donc de trouver un moyen de contourner ces problèmes, pour en faire des avantages.

Premièrement, comme expliqué dans la partie I.2.3, il est possible d'utiliser les disques virtuels comme de vrais disques-dur, que l'on peut monter dans la machine virtuelle, pour que le disque apparaisse dans l'arborescence des fichiers de Linux, de la même façon qu'une clé USB apparait dans le "Poste de travail" sur Windows.

De ce principe, il serait intéressant d'installer le système d'exploitation sur un disque virtuel, puis l'ensemble des outils utilisés sur des disques virtuels distincts.

Ainsi, le déploiement de l'ensemble de la solution ne s'effectuera qu'une seule fois, puis à chaque mise à jour ou ajout d'un outil, seul son disque dur virtuel (de "petite" taille) sera déployé.

Aux difficultés d'utilisation de la machine virtuelle, viennent s'ajouter les actions de montage et démontage des disques virtuels.

Pour obtenir une solution simple d'utilisation, il est possible d'automatiser les phases de paramétrisation et de contrôle de la machine virtuelle. De plus, une fois la solution fonctionnelle, une interface graphique permettrait à l'utilisateur d'effectuer les actions de manière simple et automatique.

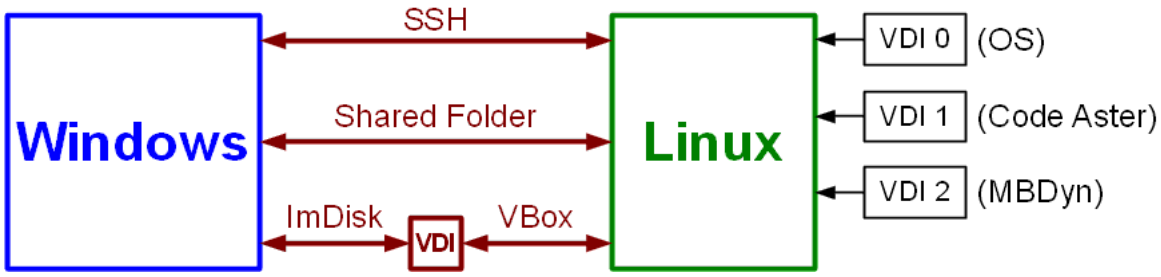

FIGURE I.10 – Schéma de la solution envisagée

La figure I.10 illustre le principe de la solution envisagée, où le système d'exploitation de la machine hôte Windows pourra échanger des fichiers avec la machine virtuelle Linux, sur laquelle on pourra y brancher différents disques virtuels et exécuter les calculs.

II Étude de fonctionnement

Avant de développer la solution, il est nécessaire d'étudier les besoins de la solution finale. Chacun de ces besoins peut mettre en œuvre l'utilisation d'outils, ou peut exiger certains réglages.

II.1 Construction des machines

II.1.1 Machine de développement

Une première machine virtuelle, composée d'une distribution Fedora 16 64 bits, a été mise en place avec un environnement complet de développement.

Cette machine comporte toutes les bibliothèques nécessaires à la compilation des différents outils.

C'est grâce à cette machine que la solution finale sera mise en place, car elle permettra de tester les différents outils utilisés.

II.1.2 Machine minimale

Une seconde machine virtuelle, a été installée de façon minimale.

Elle se base sur le même système d'exploitation (Fedora 16 64 bits) pour que les différents outils compilés sur la machine virtuelle "complète" soient compatibles avec la machine virtuelle "minimale".

Les avantages d'un système d'exploitation minimal, sont :

- démarrage et arrêt plus rapide : Lors du démarrage le système démarre les différents services qui mettent un certain temps à s'exécuter. Ici, seuls les services vitaux et ceux que l'on utilisera seront démarrés. De plus, lors de l'extinction le système aura moins de processus à arrêter.
- système moins "lourd" : Lorsque le système est en marche, de nombreux processus sont lancés, et utilisent des ressources (CPU et RAM).

– absence d'interface graphique : Le serveur X qui permet l'affichage des différentes fenêtres graphiques consomme aussi des ressources.
– taille minimale : Seuls les programmes indispensables sont installés, ce qui permet d'avoir une installation minimale (200Mo au lieu des 4Go).

Cette machine n'est contrôlable uniquement en ligne de commande, ce qui est assez compliqué pour effectuer les différents tests. On utilisera donc cette machine virtuelle une fois la solution fonctionnelle.

II.2 Paramétrisation de la machine virtuelle

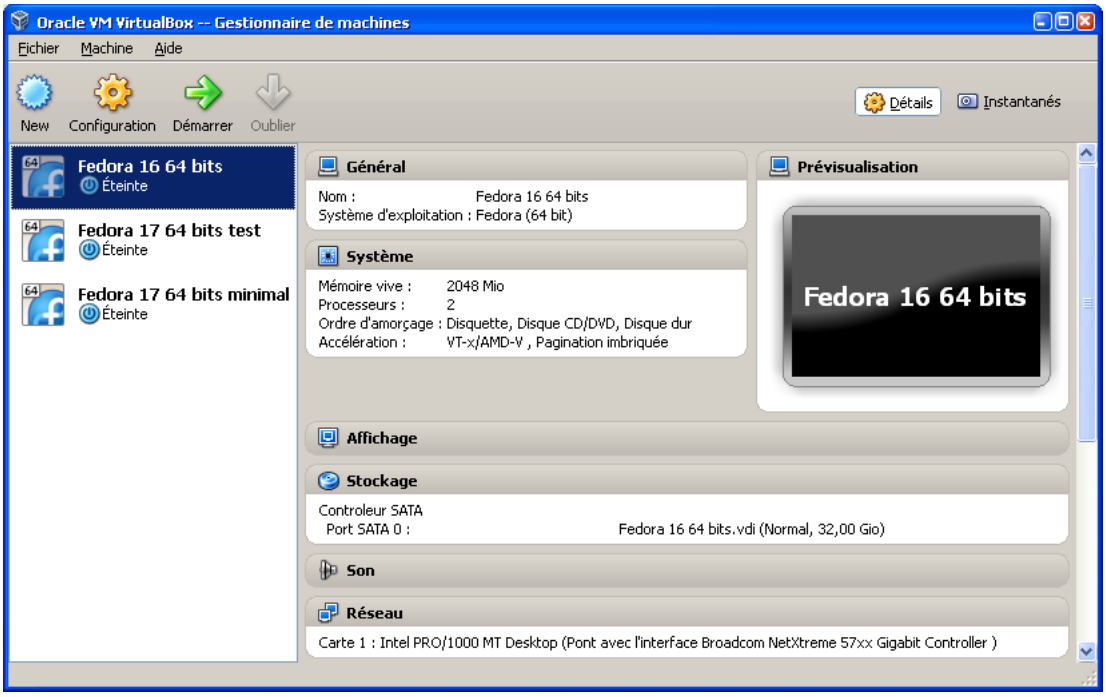

FIGURE II.1 – Interface graphique de VirtualBox

VirtualBox dispose d'une interface graphique, présente sur la capture d'écran II.1, qui permet de paramétrer facilement l'ensemble des machines virtuelles.

Cette interface possède toutefois quelques limites, car lorsque la machine virtuelle est allumée, il est impossible de modifier la majorité de ses paramètres.

L'objectif du projet est de fournir une solution de virtualisation simple, qui permettrait d'effectuer des calculs scientifiques de manière automatique. Il faut donc trouver une solution

efficace qui permettrait de paramétrer automatiquement l'interface graphique, pour que l'utilisateur n'ait pas à le faire.

II.2.1 LibVirt

L'utilisation de deux types différents de machine virtuelle (VirtualBox et VMware) dans l'entreprise Hutchinson impose le développement d'une solution compatible avec chacune d'elles.

LibVirt[1] est une API[2] de gestion de virtualisation, qui permet de contrôler de nombreuses plateformes de virtualisation, telles que :

- VirtualBox ;
- QEMU ;
- VMware ...

C'est une bibliothèque open-source, codée en Langage C et développée par la société Red Hat, qui utilise une couche d'abstraction pour être compatible avec chacune d'entre elles.

Initialement prévue pour être utilisée sur une plateforme Unix/Linux, il serait possible de l'installer et l'utiliser sur les différents systèmes d'exploitation, dont Windows, en effectuant quelques manipulations. La première méthode consiste à compiler la bibliothèque avec MinGW, qui est un émulateur de terminal Linux qui permet la compilation de programmes Linux en exécutables Windows. Rencontrant plusieurs problèmes, notamment en raison de problèmes de certificats de sécurité, cette première méthode n'a pu aboutir.

La seconde méthode a pour objectif de cross-compiler LibVirt. Cela consiste à compiler la bibliothèque sous Linux, en spécifiant certaines options lors de la compilation, pour pouvoir être exporté sur un système d'exploitation différent. Dans le cas de l'exportation de Linux vers Windows, la compilation produira des exécutables (d'extension *.exe*) et des bibliothèques dynamiques (d'extension *.dll*). A la suite de cette cross-compilation les fichiers générés seraient directement exécutables sous Windows. Cette méthode a été concluante car il était bien possible d'utiliser LibVirt dans des scripts Python, mais la connexion avec la machine virtuelle était impossible pour des raisons inconnues.

II.2.2 VBoxManage

Suite à l'incapacité d'utiliser la bibliothèque LibVirt, il fallait donc trouver une autre technique pour contrôler la machine virtuelle.

1. LibVirt - Site web : http ://libvirt.org
2. API (Application Programming Interface) : ensemble de fonctions mises à disposition par une bibliothèque logicielle, permettant l'interaction entre les différents programmes.

VirtualBox propose, en plus de l'interface graphique présentée en début de la partie II.2, un programme en ligne de commande : `VBoxManage`. Il propose la configuration des machines virtuelles (nombre de cœurs et RAM allouée, ...), mais aussi la manipulation et l'affichage d'informations des disques virtuels, des répertoires partagés, ...

Ce programme est installé en même temps que le logiciel VirtualBox, et peut être directement utilisé dans les terminaux de la machine hôte.

II.2.3 VirtualBox SDK

VirtualBox propose un SDK [3] permettant aussi un contrôle des machines virtuelles. Cette API propose plusieurs classes, programmées dans plusieurs langages : C++ et Python.

Contrairement à VBoxManage, ce SDK n'est pas installé avec VirtualBox. Il faut donc télécharger [4] puis l'intégrer à la solution.

II.2.4 Méthode choisie

Ne réussissant pas à utiliser LibVirt, la solution devra se résoudre aux machines virtuelles de type VirtualBox.

Pour développer ma première solution, j'utiliserai VBoxManage, qui permet un contrôle simple et efficace des machines virtuelles en ligne de commande. L'API pourrait être utilisée dans une version future, pour optimiser ou compléter la première solution.

II.3 Contrôle de la machine virtuelle

II.3.1 Nécessité de communiquer avec la machine virtuelle

Le programme VBoxManage ne permet que de paramétrer la machine virtuelle (nombre de cœurs alloués, quantité de RAM, manipulation de disques virtuels, ...), mais ne permet aucune action à l'intérieur du système d'exploitation virtualisé. Par exemple, il est impossible d'accéder à l'arborescence des fichiers, de manipuler les répertoires, d'exécuter un outil installé dans la machine virtuelle, ...

3. SDK (Software Development Kit) : ensemble d'outils mis à la disposition des développeurs, permettant la création d'application.
4. SDK VirtualBox - Source : https ://www.virtualbox.org/wiki/Downloads

II.3.2 Configuration de la machine hôte

II.3.2.1 PuTTY

PuTTY[5] est un client SSH, gratuit et open-source, initialement développé pour Windows mais portable sur Linux, qui propose aussi d'autres protocoles de communication tels que Telnet, TCP ou Rlogin. Il possède une interface graphique qui facilite la configuration du logiciel avant la connexion.

PuTTY est très utilisés sous Windows car il dispose de nombreuses fonctionnalités. De plus, il est installé sur tous les ordinateurs du service d'Hutchinson, ce qui permet de ne pas devoir installer de logiciels supplémentaires pour déployer la solution de virtualisation.

II.3.2.2 PLink

PuTTY propose de nombreux outils liés à la communication par protocole SSH, dont PLink (PuTTY Link). Il s'agit d'un client SSH, Telnet et Rlogin, exécutable en ligne de commande, qui permet de se connecter sur un serveur distant, ou d'y lancer des commandes à distance.

II.3.3 Configuration de la machine virtuelle

II.3.3.1 Serveur SSH

Il est tout d'abord nécessaire d'installer un serveur SSH sur la machine virtuelle, pour permettre la communication entre la machine hôte et la machine virtuelle, plus précisément entre le client (PLink installé sur la machine hôte) et le serveur (installé dans la machine virtuelle).

OpenSSH est un logiciel libre et open-source, fournissant un ensemble d'outils liés à la communication par protocole SSH. Il s'installe facilement en utilisant le gestionnaire de paquet (expliqué plus tard dans la partie III.5.1.3) disponible sur Linux, `yum` ou `apt-get` selon la distribution utilisée :

```
# yum install openssh-server
# apt-get install openssh-server
```

Une fois installé le démon (expliqué en annexe dans la partie 1.3.2) SSH (sshd) se lancera automatiquement à chaque démarrage de la machine virtuelle.

5. PuTTy - Site web : http ://www.chiark.greenend.org.uk/∼sgtatham/putty

II.3.3.2 Droits administrateur

La manipulation de la machine virtuelle impose d'effectuer des actions "critiques" au sein du système d'exploitation Linux, comme par exemple le montage et démontage de disques (commandes `mount` et `umount`), la création et suppression de répertoires ou fichiers (commande `mkdir` et `rm`). Par défaut, les utilisateurs enregistrés sur Linux ne possèdent pas les droits administrateur, pour des raisons de sécurité, et ne peuvent pas exécuter ces commandes ou leur applications sont limitées.

La commande `sudo` (Subsitute User DO) permet aux utilisateurs, autorisés par l'administrateur, de lancer certaines commandes en tant que super-utilisateur [6] :

```
$ sudo commande
```

Pour autoriser un utilisateur à exécuter ces commandes l'administrateur système doit lui ajouter les privilèges. Pour cela il doit modifier le fichier de configuration `/etc/sudoers`, et ajouter la ligne suivante :

```
username  ALL=cmd1, cmd2, NOPASSWD: cmd3, cmd4
```

où

username est le nom de l'utilisateur qui recevra les droits

cmd1 et cmd2 sont les commandes qui pourront être exécutées par l'utilisateur. Il est possible d'autoriser toutes les commandes en spécifiant `(ALL)`

cmd3 et cmd4 sont les commandes qui ne nécessiteront pas du mot de passe de l'utilisateur. Il est aussi possible d'ignorer la demande du mot de passe pour toutes les commandes en spécifiant `(ALL)`.

Ainsi pour autoriser l'utilisateur à exécuter toutes les commandes en tant que super-utilisateur sans demande de mot de passe, il suffit d'entrer la ligne :

```
username  ALL=(ALL) NOPASSWD: ALL
```

Notons que la modification du fichier `/etc/sudoers` soit s'effectuer grâce à la commande `visudo` qui permet une vérification du fichier avant enregistrement et le chargement des nouveaux paramètres dans le système.

Enfin, une option est à désactiver dans le fichier `/etc/sudoers` :

```
Defaults requiretty
```

Si cette option n'est pas désactivée, les requêtes envoyées par PLink n'aboutiront pas, et retourneront un message d'erreur :

```
sudo: sorry, you must have a tty to run sudo
```

Une fois désactivée, par exemple en commentant la ligne avec un "#", il est possible d'utiliser la commande `sudo` à partir d'un ordinateur distant et sans demande de mot de passe.

6. Super-utilisateur (ou root) : utilisateur qui possède toutes les permissions.

II.3.3.3 La carte réseau

Pour pouvoir communiquer directement avec la machine virtuelle il est nécessaire de paramétrer le réseau dans le type "Accès par pont", comme expliqué dans la partie I.2.3.3. Ainsi la machine virtuelle possèdera sa propre adresse IP et pourra être accessible via l'outil PLink.

Il est nécessaire de connaitre l'adresse IP de la machine virtuelle pour y envoyer des commandes. L'adressage IP sera donc réglé en "IP statique" avec une adresse IP spécifique plutôt qu'en "IP dynamique", pour ne pas avoir à retrouver l'IP après chaque démarrage de la machine virtuelle.

II.3.4 Utilisation de PLink

PLink s'exécute en ligne de commande, et requiert de spécifier les options de connexion au serveur cible :

```
plink <options> "<command>"
plink -ssh -l <username> -pw <password> <IP> "<command>"
```

Après exécution de la commande, le serveur cible renvoie au client l'affichage de retour de la commande.

Ce mode de communication est très efficace, mais a une latence assez élevée. En effet, il faut environ 1/2 seconde au minimum pour envoyer une simple commande au serveur. Donc lorsqu'il s'agit d'exécuter un grand nombre de commandes, l'envoi des commandes les unes après les autres peut durer un certain temps.

La solution consiste à envoyer plusieurs commandes simultanément au serveur, en les séparant par des ";". Elles seront exécutées les unes à la suite des autres, et l'affichage de chacune des commandes sera retourné :

```
plink <options> "<command1> ; <command2> ; <command3>"
```

Ainsi, l'envoi d'un programme Bash [7] permettra d'exécuter un mini-programme (avec tests et conditions) sur la machine virtuelle.

II.4 Les échanges de fichiers

Une des nécessités dont doit se doter la solution, est la possibilité d'échange de fichiers entre la machine hôte (Windows) et la machine virtuelle (Linux). En effet, pour pouvoir

7. Bash (Bourne-Again SHell) : shell du projet GNU, qui est l'interpréteur permettant d'exécuter des commandes.

effectuer des calculs, les outils doivent disposer d'un fichier d'entrée, et vont générer des fichiers de sortie. Il faut donc pouvoir envoyer le fichier d'entrée, puis récupérer les fichiers de sortie.

II.4.1 Le répertoire partagé

Cette méthode utilise la fonction du répertoire partagé : accéder aux fichiers de la machine hôte à partir de la machine virtuelle.

II.4.1.1 Partager un répertoire

Le partage d'un répertoire peut s'effectuer en utilisant l'interface graphique proposée par VirtualBox, mais est aussi possible en ligne de commande en utilisant le programme VBoxManage :

```
vboxmanage sharedfolder add <VM_NAME> --name <LABEL>
                           --hostpath <PATH>
```

où

<VM_NAME> : est le nom de la machine virtuelle avec qui l'on souhaite partager le répertoire,

<LABEL> : est le nom unique qui sera utilisé pour identifier le répertoire partagé,

<PATH> : est le chemin du répertoire, de la machine hôte, que l'on souhaite partager.

Ensuite il est nécessaire de monter le répertoire dans la machine virtuelle, pour qu'il soit accessible dans l'arborescence Linux. Contrairement aux périphériques, le répertoire partagé n'est pas représenté par un fichier (explications faites dans la partie III.5.4.1). Cela s'effectue donc en utilisant son *label* :

```
# mount -t vboxsf <LABEL> <REP_MONT>
# mount.vboxsf <LABEL> <REP_MONT>
```

où `mount.vboxsf` est un alias pour `mount -t vboxsf`.

II.4.1.2 Principe de l'échange

L'émission : Prenons l'exemple d'un fichier source `C:\chemin...source\fichier.txt` que l'on souhaite envoyer vers le répertoire `/chemin...cible/repertoire/` :

1. on partage le répertoire parent de la cible avec la machine virtuelle : `C:\chemin...source\`
2. on monte le répertoire partagé dans la machine virtuelle, à un endroit "quelconque", par exemple : `/media/`
3. on effectue (dans la machine virtuelle) la copie du fichier `/media/fichier.txt` vers le répertoire cible `/chemin...cible/repertoire/`

4. on démonte le répertoire partagé (optionnel)

Cette méthode fonctionne aussi pour l'envoyer d'un répertoire.

La réception : Pour recevoir un fichier `/chemin...source/fichier.txt` vers le répertoire `C:\chemin...cible\`, la méthode est quasiment la même :

1. on partage le répertoire cible avec la machine virtuelle : `C:\chemin...cible\`
2. on monte le répertoire partagé dans la machine virtuelle, à un endroit "quelconque", par exemple : `/media/`
3. on effectue (dans la machine virtuelle) la copie du fichier `/chemin...source/fichier.txt` versle répertoire cible `/media/`
4. on démonte le répertoire partagé (optionnel)

Cette méthode fonctionne aussi pour la réception d'un répertoire.

II.4.2 SSH

Le protocole SSH permet, en plus de l'envoi de commande, d'échanger des fichiers très simplement en ligne de commande, et de façon sécurisée.

II.4.2.1 Configuration de la machine hôte

PuTTY propose un autre outil, disponible aussi en ligne de commande, qui permet l'échange de fichier avec un serveur distant : *Pscp* (PuTTY Secure Copy). Il est, comme PLink, installé par défaut avec le logiciel PuTTY (confère partie II.3.2.2 pour plus de détails). Ce programme s'exécute en ligne de commande, et permet l'utilisation de différents protocoles d'échange : SSH, FTP, ...

II.4.2.2 Configuration de la machine virtuelle

La machine virtuelle doit disposer d'un serveur SSH pour que le client (Plink) s'y connecte. La configuration sera exactement la même que celle utilisée pour l'envoi de commande, développée dans la partie II.3.3. Aucune configuration supplémentaire ne sera donc nécessaire pour permettre l'échange de fichiers.

II.4.2.3 Utilisation

L'envoi et la réception de données s'effectuent de manière très simple et requiert, comme avec PLink, les différentes options pour permettre l'échange avec le serveur.

L'envoi de fichier ou de répertoire s'effectue de cette manière :

```
pscp -r -pw <password> <source> <username>@<IP>:<target>
```

et la réception de manière similaire :

```
pscp -r -pw <password> <username>@<IP>:<source> <target>
```

II.4.3 Disque virtuel

II.4.3.1 Principe

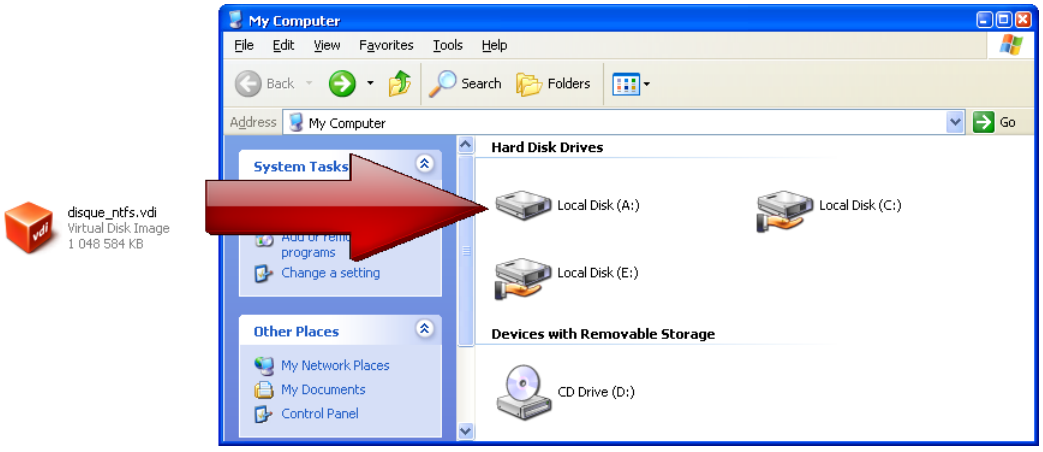

FIGURE II.2 – Montage d'un disque virtuel dans la machine hôte (Windows)

Le disque virtuel permet, grâce à l'hyperviseur, d'émuler un disque dur (ou une clé USB). Comme expliqué dans la partie I.2.3, ils peuvent être branchés ou débranchés de la machine virtuelle comme s'ils étaient de vrais disques durs.

Mais il est aussi possible d'utiliser ce disque virtuel de la même manière sur la machine hôte Windows. Il existe des outils qui permettent de monter ce disque virtuel pour qu'il apparaisse en tant que nouveau volume, dans l'interface du "Poste de travail". Cela permet ainsi de parcourir les fichiers, de les modifier, et même de formater le disque comme s'il s'agissait d'un vrai disque dur.

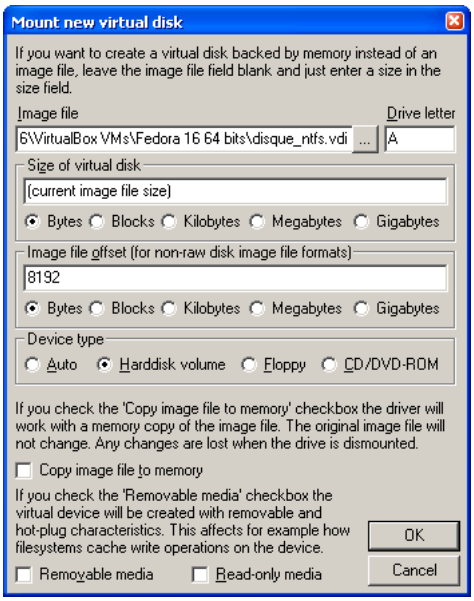

FIGURE II.3 – Interface graphique de imDisk

II.4.3.2 imDisk

Pour ce faire, j'utiliserai *imDisk*[8] qui est gratuit et open-source. Ce programme dispose d'une interface graphique (capture d'écran II.3), mais est aussi utilisable en ligne de commande.

Pour monter un disque virtuel sur Windows il est nécessaire de spécifier un *offset*. Cet offset correspond à la taille des données situées en en-tête du fichier (confère la partie I.2.3.2 qui détaille la structure d'un disque virtuel). Ainsi le programme accèdera directement au disque encapsulé.

II.4.3.3 Principe de l'échange

L'échange de fichier peut donc s'effectuer par l'intermédiaire d'un disque virtuel de la même manière que l'on échangerait un fichier entre deux ordinateurs à l'aide d'une clé USB :

1. monter le disque virtuel sur la machine hôte (Windows) ;
2. copier les fichiers sources sur le volume ;
3. démonter le disque virtuel de la machine hôte ;
4. brancher puis monter le disque virtuel dans la machine virtuelle (Linux) ;
5. copier les fichiers vers le répertoire cible ;
6. démonter puis débrancher le disque virtuel de la machine virtuelle.

8. imDisk - Lien : http ://www.ltr-data.se/opencode.html/#ImDisk

Pour la réception de fichiers le principe est exactement le même mais en copiant d'abord la source dans la machine virtuelle avant d'effectuer la copie dans la machine hôte.

II.4.3.4 Inconvénients

Cette solution fonctionne, mais pose plusieurs problèmes :

- Cela demande de nombreuses opérations qui sont susceptibles d'échouer : disque virtuel inexistant, corrompu ou de trop petite taille, échec du branchement, débranchement du disque alors que la copie n'est pas terminée, ...
- Ensuite, chacune de ces opérations peut durer quelques secondes. Par exemple lorsque l'on branche un disque virtuel sur la machine virtuelle (Linux), celle-ci ne le reconnait pas instantanément et il faut attendre environ 2 secondes.
- Enfin l'échange impose d'effectuer la copie deux fois : une première fois sur le disque virtuel, et une seconde fois dans la machine virtuelle (si envoi) ou hôte (si réception).

II.4.4 Comparaison

Deux types de calcules sont généralement effectués sur les outils scientifiques :

- Le calcul intensif qui s'effectue sur un fichier d'entrée volumineux pouvant atteindre 1Go ;
- Le plan d'expérience qui s'effectue quant à lui sur une multitude de petits fichiers d'entrée.

Pour pouvoir comparer ces différentes solutions, j'ai étudié la rapidité de l'échange en faisant varier le nombre de fichiers, ainsi que leur taille.

II.4.4.1 Conditions des tests

Pour ne pas influencer les résultats, ni la machine hôte ni la machine virtuelle n'ont été utilisées lors des différents tests. La fonction `time.time()` a été introduite dans les méthodes d'échange, permettant d'obtenir l'heure courant, et donc obtenir la durée de l'échange par différence. Les résultats sont obtenus en calculant la moyenne sur un échantillon de 5 échanges, le nombre de fichier varie entre 1 et 10.000, et la taille du fichier varie entre 1ko et 1Go.

II.4.4.2 Résultats obtenus : analyse et critique

Notons que l'axe des ordonnées se limite à 140 secondes, mais le temps d'échange avec la méthode SSH atteint 1250 secondes pour 10.000 fichiers. Les résultats sont reportés dans un tableau en annexe.

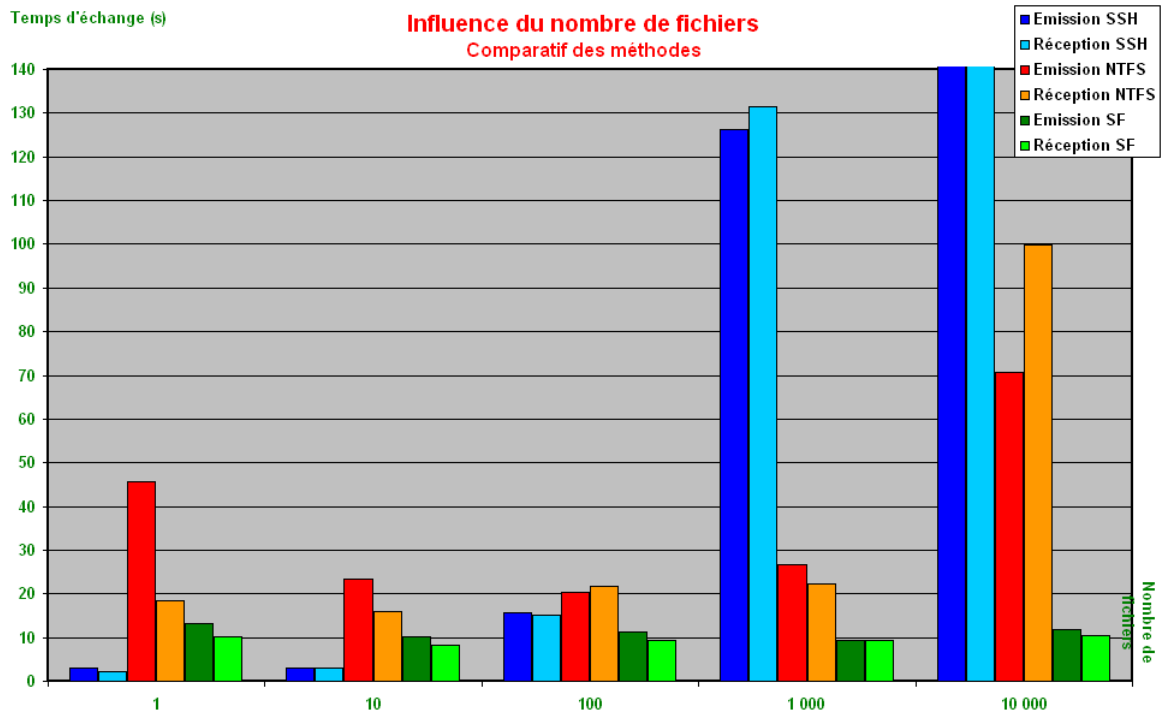

FIGURE II.4 – Temps d'échange en fonction du nombre de fichiers

SSH : On peut remarquer que le temps d'échange est "plus ou moins" proportionnel au nombre de fichier. De plus, le temps d'échange est relativement faible et constant, pour des fichiers de taille inférieure à 10Mo et pour un ensemble de 100 fichiers.

Disque virtuel : Le graphique "Étude de la méthode NTFS" présent en annexe détaille les différentes étapes de l'échange. La durée notée "Autre" correspond aux étapes de (dé)branchement et (dé)montage du disque virtuel dans la machine virtuelle et la machine hôte. Ce temps est constant car les fichiers n'ont aucune influence sur ces actions.

On remarque que les inconvénients présentés dans la partie II.4.3.4 sont bien présents :

- la durée des deux copies est globalement très faible et constante, mais se fait doublement ressentir lors de la réception de 10.000 fichier ou d'un fichier de 100Mo,
- la durée de l'échange est très élevée pour les fichiers de petite taille et en petit nombre,
- un problème survient lors de la copie d'un fichier d'1Go sur le disque virtuel monté dans Windows, apparemment lors du "flush du buffeur".

Répertoire partagé : Le graphique "Étude de la méthode SF" présent en annexe détaille les différentes étapes de l'échange. La durée notée "Autre" correspond aux étapes de partage du répertoire et de (dé)montage du répertoire partagé dans la machine virtuelle. De même que pour le disque virtuel, la durée est constante car les fichiers n'ont pas d'influence sur les actions.

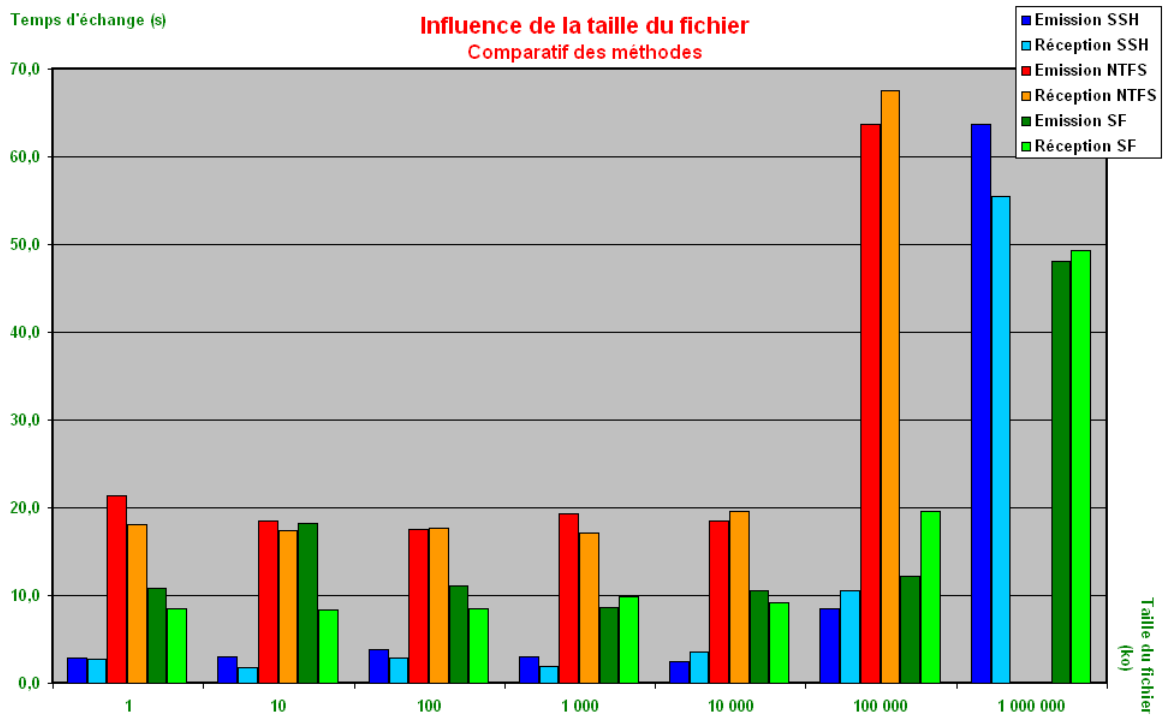

FIGURE II.5 – Temps d'échange fonction de la taille du fichier

Cette méthode est globalement la plus performante, car le temps de l'échange est constant quelque soit le nombre et la taille des fichiers, à l'exception du fichier de 1Go où la durée de l'échange atteint raisonnablement les 50 secondes.

Il serait possible d'améliorer le temps noté "Autre" en développant un script qui monterait et effectuerait la copie (étape 2 et 3), plutôt que d'exécuter les deux commandes séparément (confère la partie II.4.1.2 pour plus de détails). Cette technique est évoquée dans la partie II.3.4 pour diminuer fortement le temps d'exécution en lançant plusieurs commandes simultanément avec PLink.

II.4.4.3 Conclusion

Pour des raisons de simplicité de programmation et d'utilisation j'utiliserai la méthode SSH dans la première version de ma solution, qui permet un temps de transfert assez rapide pour des fichiers de taille et quantités raisonnables.

La solution qui pourrait être utilisée dans les prochaines versions serait d'effectuer un test sur la source à échanger. Par exemple si le nombre ou la taille des fichiers à échange est faible on utilisera la méthode SSH, et pour les gros ou nombreux fichiers on utilisera le répertoire partagé.

III Réalisation de la solution

Une fois le problème étudié, et les techniques qui seront utilisées définies, il faut développer la solution pour automatiser le fonctionnement de la machine virtuelle.

III.1 Langage utilisé

III.1.1 Qu'est-ce que Python ?

Python[1] est un langage de programmation récent, créé en 1989 par Guido van Rossum. Il est codé en Langage C, open-source et gratuit. De plus, Python est orienté objet, favorisant la programmation impérative structurée, et possède un type dynamique. Python est *multiplate-forme*, c'est à dire qu'il peut être porté aussi bien sur Windows Linux ou Mac OS, mais aussi sur cluster, sans aucune adaptation.

Contrairement aux langages compilés, comme le Langage C ou le C++, Python est un langage *interprété*. Un programme intermédiaire, appelé *interpréteur*, a pour but d'analyser, de traduire, puis d'exécuter le code source ligne par ligne, qui n'est pas directement exécutable par la machine.

La machine virtuelle Python permet, comme pour Java ou C#, de bénéficier du *Ramasse-miettes*, aussi appelé *garbage collector* en anglais, qui permet la gestion de la mémoire. Lors de l'exécution du programme, celui-ci va allouer de mémoire au fur et à mesure de son exécution. Lorsque la mémoire allouée n'est plus utilisée, c'est à dire qu'aucun objet n'a de référence sur celle-ci, le garbage collector va libérer la mémoire qui pourra être utilisée par la suite.

C'est un langage de *haut niveau* disposant de nombreux outils, appelés *module*, mis à disposition des programmeurs pour faciliter le développement d'application, dont voici quelques exemples :

os : pour l'interaction avec le système d'exploitation ;

re : pour l'utilisation des expressions régulières ;

unittest : permettant d'effectuer des tests unitaires ;

1. Site web : http ://www.python.org

threading : pour l'utilisation de threads.
Ils sont comparables aux "packages" du langage Java, ou des "namespace" du C++.

De plus, sa syntaxe est épurée et concise, pour simplifier son utilisation : l'indentation du code source a remplacé les caractères spéciaux, utilisés dans la majorité des langages pour imbriquer les différentes parties du code.

III.1.2 Choix de Python

Python est un langage de plus en plus utilisé dans le monde. Il dispose de mises-à-jour régulières qui permettent d'améliorer ses performances et sa stabilité, avec l'ajout de fonctionnalités qui le rend très complet.

Ce langage a été choisi par Hutchinson pour le développement de nombreux logiciels car il offre de nombreux modules scientifiques très utilisés dans le centre de recherche, comme par exemple :

- NumPy : destiné à la manipulation de matrices et de tableaux multidimensionnels ;
- matplotlib : permettant le tracé de graphiques 2D ou 3D.

De plus, il permet de s'interfacer avec de nombreux langages de programmation, comme le Fortran ou le C++ très utilisés pour le développements d'applications scientifiques.

De plus, Python est utilisé sur le cluster de calcul de l'entreprise, donc l'intégration de programme Python devient alors très facile.

Le projet sera donc développé dans ce langage, en utilisant les différents modules disponibles et utilisés par Hutchinson.

III.2 Les tests unitaires

Un des inconvénients du langage interprété est le fait qu'il n'analyse que les lignes de codes au moment de l'exécution. De plus, le typage dynamique peut lever des erreurs. De ce fait, il devient difficile de déceler les erreurs situées dans les portions de code peu appelées (notamment la gestion des cas particuliers), comme par exemple un nom de fonction invalide, un argument manquant, ou une variable inexistante.

Les tests unitaires permettent de vérifier le bon fonctionnement d'un programme, ou d'une de ses parties. Cela consiste à s'assurer que le programme a le comportement attendu, quelque soit la situation dans lequel il se trouve. Les tests doivent être effectués de manière isolée (ne pas dépendre des autres fonctions non-testées), et automatique.

Par exemple, lors d'une copie de fichier, il faut vérifier que la fonction (ou méthode) appelée lèvera bien une exception lorsque :

- le fichier d'entrée est inexistant ;
- le répertoire cible est inexistant ;
- les droits d'accès sont insuffisants ;
- la copie est un échec.

Pour minimiser les risques d'erreurs, des tests unitaires ont été développés, permettant de vérifier le bon comportement de l'application dans les différents cas particuliers. La "bonne pratique" des tests unitaires impose d'utiliser un fichier par classe testée, contenant une classe par méthode, et ayant une méthode par cas particulier.

J'ai utilisé le module *PyUnit*, intégrant de nombreuses classes et méthodes permettant d'effectuer des tests unitaires de manière simple et rapide. Cela m'a permis de corriger de nombreux bugs, notamment dans l'utilisation des scripts Shell envoyés à la machine virtuelle.

III.3 La documentation

Documenter le code source est une pratique très importe dans le développement d'une application. Cela peut s'avérer extrêmement pratique pour la maintenance d'un programme, car le développeur explique son fonctionnement, mais cela devient indispensable pour les utilisateurs qui utiliseront par exemple une bibliothèque existante. On dit parfois qu'« un bon code source doit être compréhensible uniquement avec les commentaires ».

L'application développée durant le stage pourrait être utilisée ou améliorée à l'avenir. Il est donc impératif de faciliter sa maintenance, en la documentant correctement.

III.3.1 Documentation Python

Python permet d'intégrer directement la documentation au sein du code source, qui pourra être obtenu dynamiquement lors de l'exécution. Cela nécessite l'écriture d'une chaine de caractères non référencée au sein de la classe, la méthode ou la fonction. La documentation pourra ensuite être appelée dynamiquement grâce à l'attribut `__doc__`.

Voici un exemple :

```
# Declaration de la fonction avec sa documentation
def maFonction() :
    " Ma documentation "
    pass
```

```
# Appel de l'attribut contenant la documentation
if __name__ == '__main__' :
    print maFonction.__doc__
```

affichera :

```
Ma documentation
```

Il existe le module *PyDoc* qui est l'outil de documentation de Python. Il permet d'afficher la documentation d'un module ou d'une fonction, de manière simple :

```
pydoc <module/>
pydoc <classe>
pydoc <methode/fonction>
```

III.3.2 Doxygen

Doxygen[2] est un programme qui permet de générer une documentation à partir du code source d'une application. De nombreux langages peuvent être analysés : Langage C, C++, Java, Python, ...La documentation peut être générée dans de nombreux formats : HTML, PDF, LaTeX, ...Doxygen est gratuit et open-source, et est compatible pour les différents systèmes d'exploitation (Windows, Linux, ...). Ce programme est le plus utilisé par les différents développeurs, du fait de ses nombreuses fonctionnalités et sa facilité d'utilisation.

Pour que le programme considère un commentaire comme un "commentaire doxygen", il est nécessaire de respecter certaines règles d'écriture, telles que l'utilisation du symbole "@" ou "\" suivi d'un mot clé, ou la délimitation du bloc de commentaire car le double symbole "**" (selon le langage de programmation utilisé).

```
/**
* @details Classe representant une coordonnee dans l'espace.
* @author Julien
*/
class Point {
    /**
    * @brief Constructeur.
    * @param x Abscisse
    * @param y Ordonnee
```

2. Doxygen - Site web : http ://www.doxygen.org

```
        */
        Point( int x, int y ) {
            ...
        }
    }
```

III.3.3 Filtre d'entrée

Doxygen nécessite d'utiliser la documentation avant la déclaration de la fonction, alors que Python requiert la documentation à l'intérieur de celle-ci. Il est donc impossible d'utiliser directement Doxygen pour générer la documentation d'un programme Python.

Il est toutefois possible d'utiliser un *filtre d'entrée* pour le programme Doxygen, qui va formater les commentaires du code source conformément aux règles imposées par Doxygen. Cela permet donc de générer la documentation de langages de programmation qui possède des règles différentes de celles imposées par Doxygen.

DoxyPy[3] est le filtre, gratuit et open-source, programmé en Python, que j'utiliserai pour générer la documentation de mon code avec Doxygen.

De cette façon, la documentation possèdera les avantages des deux méthodes : la documentation peut être utilisable dynamiquement en utilisant l'attribut spécial, en ligne de commande grâce à PyDoc, ou bien être exportable hors du code-source avec Doxygen.

III.4 L'interface graphique

L'interface graphique est la méthode la plus efficace pour un utilisateur de s'interfacer avec un programme. Si l'on prend l'exemple de VirtualBox, il est beaucoup plus facile d'utiliser son interface graphique (capture d'écran II.1), que d'utiliser le programme en ligne de commande VBoxManage (partie II.2.2).

III.4.1 wxPython

Comme la solution a été développée dans le langage Python, l'interface graphique sera donc développée dans ce langage, pour faciliter au maximum son intégration.

3. DoxyPy - Source : http ://pypi.python.org/pypi/doxypy/0.3

Il existe différents modules permettant de développer une interface graphique, dont les quatre plus utilisés sont :

- Tkinter : pour Tk ;
- pyGTK : pour GTK+ ;
- pyQT : pour Qt ;
- wxPython : pour wxWidgets.

J'utiliserai wxPython, car c'est le module utilisé dans le centre de recherche informatique pour développer les interfaces graphiques en Python. De plus, il permet d'intégrer d'autres modules de traçage de courbes 2D ou 3D.

Le module *wxPython*[4] est libre, multiplateforme. Il implémente wxWidgets (anciennement wxWindows), qui est aussi une bibliothèque graphique libre, multiplateforme et développée en C++. Contrairement aux bibliothèques qui restituent la même interface, wxWidgets (et donc wxPython) utilisera l'apparence native de l'environnement sur lequel il est exécuté.

Comme pour Python, wxPython permet une programmation et une gestion des événements simplifiée. Le développement repose sur l'utilisation de *sizers*, une sorte de tableau permettant le positionnement des objets de manière ordonnée les uns par rapport aux autres dans la fenêtre.

III.4.2 wxFormBuilder

III.4.2.1 Présentation

wxFormBuilder[5] est un logiciel open-source, permettant de générer facilement et rapidement le code source d'une interface graphique. Il est possible de générer une interface Python, qui utilisera la bibliothèque wxPython, mais aussi C++ ou XRC qui utiliseront la bibliothèque wxWindows.

III.4.2.2 Fonctionnement

L'environnement de wxFormBuilder, présenté sur la capture d'écran III.1, est composé de 4 grandes parties :

1. Object Tree : l'arbre des objets qui présente la disposition des différents éléments les uns dans les autres, notamment avec l'utilisation des Sizers ;
2. Component Palette : l'ensemble des composants, répertoriés dans leur catégorie. La sélection ;

4. wxPython - Site web : http ://wxpython.org
5. wxFormBuilder - Site web : https ://launchpad.net/∼wxformbuilder

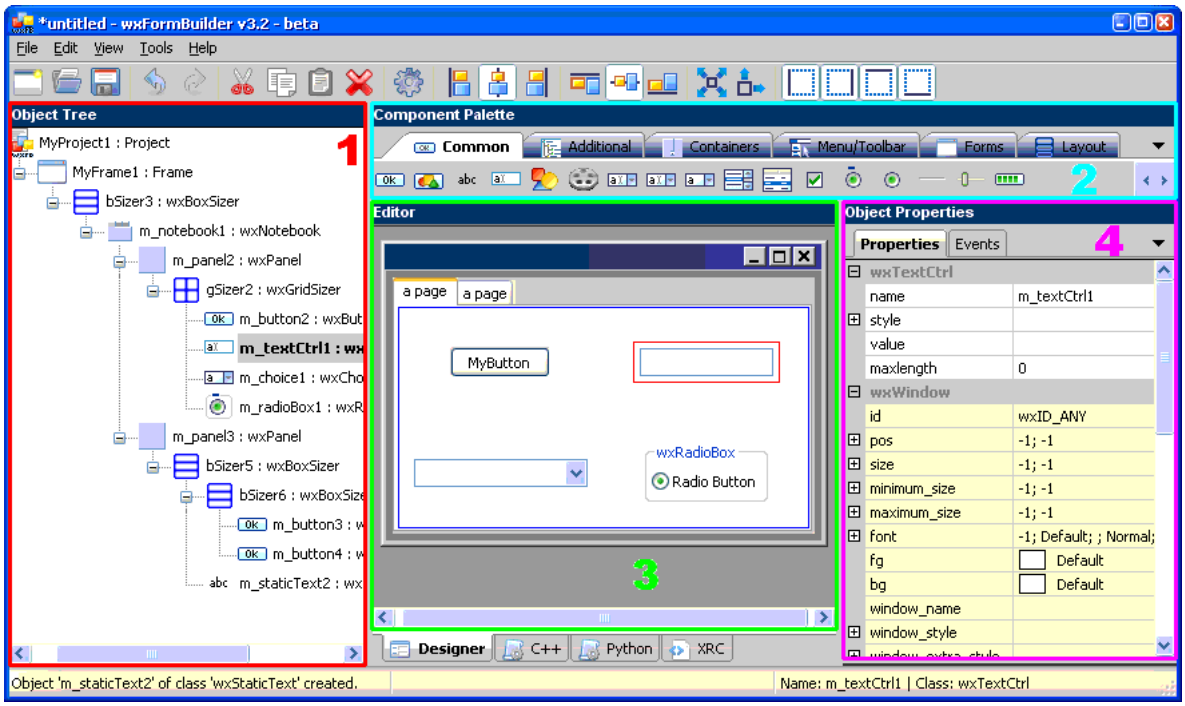

FIGURE III.1 – Interface graphique de wxFormBuilder

3. Editor : l'interface graphique générée, qui permet d'avoir un aperçu de la solution ;
4. Object Properties : les propriétés de l'objet sélectionné, tels que la taille, la position, son comportement dans le sizer, ...

Contrairement à la majorité des éditeurs graphiques, où l'on déplace des composants sur la fenêtre avant de les mettre en forme (taille, position, ...), ici la disposition des éléments est définie (uniquement) à l'aide des "layout" qui permettent de structurer la présentation.

III.4.2.3 Le code généré

Une des particularités de cet outil, est que le code source généré n'est pas directement exécutable. Il va générer un fichier (.py, .cpp/.hpp, et/ou .xrc) contenant une (ou plusieurs) classe, qui héritera [6] d'un objet de base de la bibliothèque. L'avantage de cette méthode est la possibilité d'utiliser cette classe dans plusieurs projets, sans avoir à dupliquer le code.

wxFormBuilder ne permet que de générer le code source, mais ne permet pas d'y intégrer du

6. Héritage : principe de la programmation orientée objet, dans lequel les classes "filles" possèderont les mêmes attributs et méthodes que la classe "mère". De plus, elles pourront redéfinir les méthodes héritées, et en définir des nouvelles qui seront spécifiques à sa classe.

code permettant l'initialisation des éléments ou la gestion des événements. L'interface graphique est alors *statique*, c'est à dire qu'on ne peut pas interagir avec elle : les contrôleurs tels que les "liste-box" ou les "combo-box" seront vides, et ceux du type "bouton" ou "slider" n'auront aucune réaction.

Pour ajouter du dynamisme dans l'application, la technique consiste à créer une nouvelle classe qui héritera de la classe générée par wxFormBuilder. Ensuite on utilise le polymorphisme d'héritage [7] pour y ajouter les différentes initialisations (dans les constructeurs), ainsi que les actions dynamiques (pour les événements).

III.4.3 Scinder l'interface

III.4.3.1 Pourquoi ?

Pour permettre une meilleure maintenance des outils scientifiques, la solution envisagée consistait à utiliser un disque virtuel par outil.

En se basant sur le même principe, j'ai décidé de programmer chaque interface relative à un outil dans des fichiers différents, et l'interface principale afficherait l'outil sélectionné par l'utilisateur de manière dynamique.

Ce choix permettra de ne pas avoir à modifier le code source de l'interface graphique de base lors de la modification d'outils. Par exemple, pour ajouter un nouvel outil dans la solution, en plus d'ajouter le disque virtuel dans lequel l'outil est installé, il faudra programmer un morceau d'interface graphique (avec les objets et la gestion des événements) dans un fichier que l'on ajoutera aux autres fichiers de la solution.

III.4.3.2 Règles imposées

Cette technique impose toutefois certaines règles à respecter aux interfaces de outils pour pouvoir être intégrées à l'interface de base. Si elles ne sont pas respectées, l'application ne fonctionnera pas.

Dans l'interface de base, l'objet *wxChoicebook* fonctionne de manière similaire aux onglets (objet *wxNotebook*), mais les différents éléments sont proposés sous forme de liste plutôt que sous forme d'onglet. Il contient un ensemble de panels [8] (objet *wxPanel*), et le choix dans la liste permet d'afficher le panel correspondant.

7. Polymorphisme d'héritage : redéfinition d'une méthode d'une classe héritée pour y ajouter des fonctionnalités ou changer son comportement.

8. Panel : objet graphique contenant des composants.

L'interface de l'outil doit donc être un objet héritant de la classe *wxPanel* pour pouvoir être intégré à l'interface de base.

Il est possible de changer dynamiquement des modules et des classe en Python. Le chargement d'un module s'effectue grâce à la fonction spéciale `__import__`, qui prend comme argument le nom du module, c'est à dire le nom du fichier sans son extension, et retourne la variable qui représente le module. Le module doit se trouver dans le *path*[9] pour être trouvé, on ajoutera donc le répertoire contenant l'ensemble des interfaces des outils dans le path. On peut ensuite appeler directement le contenu du module à partir de sa variable, à condition que la classe, la fonction ou la variable globale existe.

```
sys.path.append( "chemin du repertoire" )
monModule = __import__( "nom du module" )
monPanel = monModule.MaClasse( arguments )
```

L'ensemble les fichiers comportera donc une classe ayant le même nom ("MyPanel" dans mon cas), ainsi que les mêmes arguments, pour pouvoir être appelé dynamiquement.

III.5 Construction des outils

Comme expliqué plus tôt dans la partie I.3.3, chaque outil sera installé sur un disque virtuel distinct. Outre l'avantage du (re)déploiement des outils, cette solution offre certains avantages mais soulève aussi des inconvénients. De plus leur installation requiert des manipulations supplémentaires.

III.5.1 Bibliothèques, paquets et installation

Une présentation du mode de fonctionnement des programmes sous Linux permettra de mieux comprendre la méthode d'installation des outils, ainsi que des problèmes rencontrés.

III.5.1.1 Les bibliothèques

Aussi appelées *libraries* en anglais, les *bibliothèques* sont des ensembles de fonctions mis à disposition des développeurs, afin de ne pas avoir à les réécrire. La quasi-totalité des programmes les utilisent et dépendent d'elles pour fonctionner.

Il existe deux types de bibliothèques :

9. Path : variable d'environnement permettant d'enregistrer l'emplacement des différents exécutables.

Les bibliothèques dynamiques (aussi appelées *partagées*) sont installées sur le système d'exploitation. Lors de l'exécution d'un programme qui utilise ce type de bibliothèques, elles seront chargées en mémoire avant leur utilisation (voir schéma III.2). Ce programme est alors appelé *programme dynamique*. L'inconvénient est qu'il est nécessaire que le système possède ces bibliothèques pour pouvoir l'exécuter.

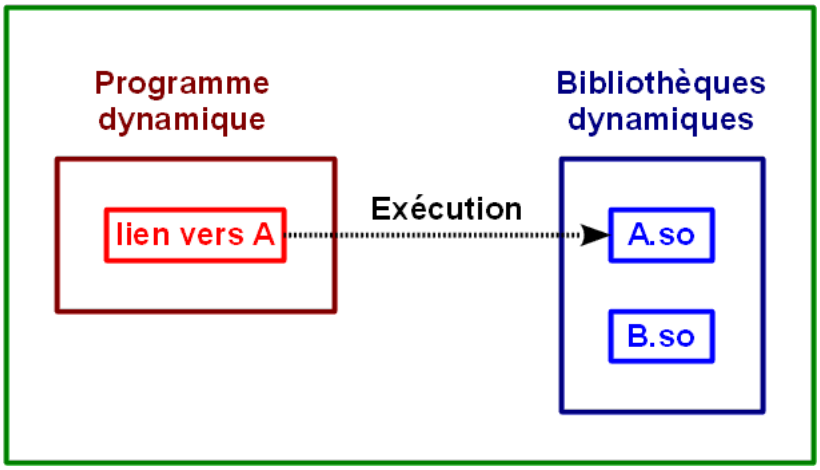

FIGURE III.2 – Schéma d'un programme dynamique

Les bibliothèques statiques sont intégrées (par copie) au programme lors de sa compilation (voir schéma III.3). Le volume de ce programme sera donc plus élevé, car la bibliothèque est copiée à l'intérieur de celui-ci. L'avantage de cette solution est que le programme sera exécutable sur les différents systèmes, même si la bibliothèque n'y est pas installée. Ce programme est alors appelé *programme statique*.

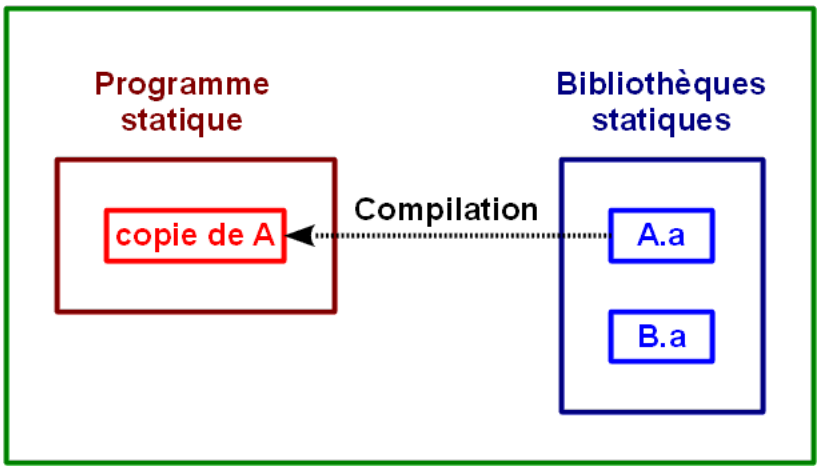

FIGURE III.3 – Schéma d'un programme statique

Notons qu'un programme peut utiliser les deux types de bibliothèques. Il sera alors dynamique, car il dépend de bibliothèques dynamiques, mais possèdera moins de dépendances.

Une bibliothèque se compose de trois parties :

1. La partie dynamique (fichiers *.dll* sous Windows et *.so* sous Unix) permet aux programmes (dynamiques) de fonctionner sur le système.
2. La partie statique (fichiers *.lib* sous Windows et *.a* sous Unix) qui est nécessaire à la compilation de programmes "statique". Il s'agit des fichiers qui seront intégrés à l'exécutable du programme pour fonctionner de manière autonome.
3. Les fichiers d'en-tête (fichiers *.h*, appelés *headers* en anglais) nécessaires à la compilation, qui contiennent les prototypes [10] des différentes fonctions de la bibliothèque.

III.5.1.2 Les paquets

Les paquets (*packages* en anglais) sont l'équivalent des exécutables de Windows (*.exe*). Il s'agit d'un type d'archives contenant les fichiers et procédures nécessaires à l'installation de programmes ou de bibliothèques.

Dans le cas d'une bibliothèque, le paquet portera le nom de la bibliothèque, précédé par *lib* (par exemple "libtool" pour la bibliothèque "tool"). Dans cette partie III.5.1, on confondra programme et bibliothèque.

Selon la distribution utilisée, ils peuvent avoir un format différent :

- Les RPM (RedHat Package Manager), d'extension *.rpm*, sont utilisés par les distributions basées sur RedHat dont Fedora ; C'est le format standard de Linux.
- Les DEB, d'extension *.deb*, sont utilisés par les systèmes basés sur la distribution Debian ;
- Les archives, d'extension *.tar.gz* ou *.gz* compressés ou non, peuvent contenir les sources de l'application ou les fichiers binaires déjà compilés. Ils sont généralement compatibles avec les différentes distributions.

Outre les différentes versions, les paquets peuvent se présenter sous diverses formes, en fonction de leur contenu :

- Les paquets dits "normaux" ne contiennent que la partie dynamique du programme nécessaire à son fonctionnement et ses dépendances. C'est la forme par défaut. ;
- Les paquets statiques (*nompaquet-static*) contiennent soit la forme autonome d'un programme, soit la partie statique d'une bibliothèque. ;

10. Prototype : syntaxe d'une fonction décrivant les paramètres d'entrée ainsi que le retour.

- Les paquets de développement (*nompaquet-devel*) contiennent l'ensemble des fichiers d'entête, permettant la compilation et la réutilisation du paquet ;
- D'autres types, plus rares comme par exemple *client* (*nompaquet-client*) et *serveur* (*nompaquet-serveur*) pour le paquet *OpenSSH* qui ne contiennent qu'une partie du programme.

III.5.1.3 L'installation

Il existe trois méthodes permettant d'installer un programme dans Linux.

Le gestionnaire de paquets est un programme permettant le téléchargement (à partir d'un dépôts [11]) et l'installation de paquets, de manière automatique et transparente. Les dépendances sont gérées automatiquement : le paquet sera installé avec les différents paquets dont il dépend, si ceux-ci ne sont pas déjà installés sur le système.

Il est utilisé en ligne de commande, mais il existe des programmes utilisant une interface graphique pour faciliter son utilisation. Sous les systèmes RedHat, sur lequel se base Fedora, le gestionnaire de paquet est `yum`, alors que sous les systèmes Debian il s'agit de `apt`.

Les paquets peuvent être manipulés "à bas niveau", c'est à dire dans l'arborescence de Linux. Les commandes `rpm` sous RedHat ou `dpkg` sous Debian, permettent l'installation, l'affichage du contenu ou des informations, ...sur un paquet. Elles permettent aussi de connaitre l'ensemble des paquets installés ou de désinstaller un paquet.

Les archives qui ne respectent pas le standard RPM ou DEB doivent être installées manuellement, en suivant le processus détaillé dans le fichier *README* ou *INSTALL*. Généralement, cela s'effectue en trois étapes :

1. Configuration des fichiers qui permettront la compilation, appelés *Makefile*, et vérification de l'état du système (bibliothèques requises installées). Cela s'effectue à l'aide du script exécutable *configure* situé à la base de l'archive.
2. Compilation des sources, transformant le code source compréhensible par l'humain exécutable par la machine.
3. Installation du programme, qui va copier les fichiers binaires dans l'arborescence de Linux.

Les commandes à exécuter sont :

11. Dépôt : serveur internet sur lequel sont répertoriés les paquets officiels ou non.

```
$ ./configure
$ make
# make install
```

C'est cette méthode qui sera utilisé pour installer les différents outils, car ceux-ci ne sont pas disponibles au format RPM.

III.5.2 Disque virtuel individuel

Le choix d'utiliser un disque virtuel par outil, expliqué dans la partie I.3.3, impose d'installer l'outil dans le disque virtuel.

III.5.2.1 Choix du répertoire d'installation

Il est possible d'installer un programme dans un répertoire spécifique plutôt que de laisser les fichiers s'installer dans les répertoires du système (*/bin*, */lib*, ...). Pour cela il existe l'option `--prefix` au script de configuration (voir partie III.5.1.3), en précisant le chemin du répertoire :

```
$ ./configure --prefix=<repertoire>
```

Ainsi les différents fichiers et exécutables du programme seront placés dans le répertoire cible : `<repertoire>/bin`, `<repertoire>/usr`, `<repertoire>/lib`, ...

Pour installer un outil sur un disque virtuel, il suffit donc de choisir comme répertoire d'installation le répertoire dans lequel est monté le disque virtuel.

III.5.2.2 Taille minimale

Une fois installés, les outils ne vont plus générer de fichier dans le répertoire d'installation, hormis les fichiers de sorties qui se trouveront dans un répertoire cible spécifié lors de l'exécution. Pour minimiser l'espace disque occupé par les différents outils sur le disque dur de la machine hôte, les disques virtuels auront une taille minimale.

Pour connaitre la taille d'une application, il faut effectuer une première installation dans un répertoire ou sur un disque virtuel temporaire. Une fois installé, la taille totale de l'outil est égale à la taille totale du répertoire d'installation.

Le disque virtuel possède un en-tête et le système des fichiers (voir partie I.2.3.2), qui représentent une certaine taille du disque virtuel. On laissera alors une marge de quelques MégaOctets sur la taille totale pour éviter tout problème lors de l'installation.

III.5.3 Compilation statique

III.5.3.1 Comparaison statique et dynamique

Avantages de la compilation dynamique :

- Les différentes bibliothèques ne sont pas dupliquées dans les différents outils, mais présentes uniquement dans la machine virtuelle ; Cela implique que la solution totale sera de taille minimale.
- Les disques virtuels seront de petite taille, mais la machine virtuelle de taille plus importante. Comme on ne déploie qu'une seule fois la machine virtuelle, et qu'il est possible de déployer plusieurs fois le même outil (mise à jour, corrections de bugs, ...), alors le déploiement sera plus rapide.

Avantages de la compilation statique :

- Il est impossible de connaitre à l'avance quelles seront les bibliothèques dont dépendront l'ensemble des outils. En effet, si un nouvel outil est ajouté à la solution, utilisant une bibliothèque non-installée, alors il sera impossible de l'exécuter ;
- L'installation des nombreuses bibliothèques dynamiques sur la machine virtuelle pose plusieurs inconvénients :
  - Bien que les paquets sont assez légers (quelques MégaOctets), la machine virtuelle possède une taille limite. Et il est impossible de modifier sa taille, sauf en cas de maintenance qui nécessiterait de redéployer la solution ;
  - Les performances du système peu décroitre avec la présence des nombreux paquets.
- Il n'est pas nécessaire de posséder une connexion internet pour télécharger les paquets.

J'ai décidé de choisir la compilation statique des outils, ce qui permet de simplifier les recherches de dépendances, malgré les disques virtuels de taille plus importante.

III.5.3.2 La compilation

Par défaut la compilation d'un programme crée un exécutable dynamique. Mais il est possible de compiler un programme de façon statique, en utilisant l'option `-static` lors de la compilation.

Comme la compilation des outils s'effectue à l'aide des Makefile, générés automatiquement, il faut que cette option soit activée dans celui-ci, et pour toutes les compilations faites. Pour cela, il existe parfois une option dans le script de configuration (confère partie III.5.1.3) permettant d'activer la compilation statique :

```
$ ./configure --disable-shared --enable-static
```

Les deux options sont complémentaires : on ne peut activer les deux en même temps. Lorsqu'une est activée (enable), l'autre est désactivée (disable).

La compilation des outils s'effectue sur la machine virtuelle complète, qui dispose de toutes les bibliothèques statiques nécessaires. En effectuant la compilation des différents outils, ceux-ci ne sont pas compilés en "statique". J'ignore l'origine du problème, car les options sont correctes et aucune erreur n'est survenue lors de l'installation.

III.5.3.3 Solution au problème

Pour connaitre l'ensemble des bibliothèques dynamiques dont dépend un exécutable, il existe la commande `ldd`. A chaque bibliothèque est associé le chemin d'installation, ou "not found" si celle-ci n'est pas installée.

```
$ ldd ./mbdyn

linux-vdso.so.1 =>  (0x00007fff20792000)
liblapack.so.3 => /usr/lib64/atlas/liblapack.so.3 (0x00007fbaaae58000)
libblas.so.3 => /usr/lib64/libblas.so.3 (0x00007fbaaac02000)
libpthread.so.0 => /lib64/libpthread.so.0 (0x0000003217800000)
librt.so.1 => /lib64/librt.so.1 (0x00000037ef600000)
libgfortran.so.3 => /usr/lib64/libgfortran.so.3 (0x00007fbaaa8eb000)
libquadmath.so.0 => /usr/lib64/libquadmath.so.0 (0x00007fbaaa6b5000)
libstdc++.so.6 => /usr/lib64/libstdc++.so.6 (0x00000036b5800000)
libm.so.6 => /lib64/libm.so.6 (0x00000036af400000)
libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00000036af800000)
libc.so.6 => /lib64/libc.so.6 (0x00000036ae400000)
libf77blas.so.3 => /usr/lib64/atlas/libf77blas.so.3 (0x00007fbaaa497000)
libcblas.so.3 => /usr/lib64/atlas/libcblas.so.3 (0x00007fbaaa276000)
/lib64/ld-linux-x86-64.so.2 (0x00000036ae000000)
libatlas.so.3 => /usr/lib64/atlas/libatlas.so.3 (0x00007fbaa9b7a000)
```

Certaines bibliothèques ne sont pas disponibles directement dans les dépôts officiels du gestionnaire de paquet. Celles-ci sont incluses dans un paquet, ayant un nom totalement différent, et sont installées en même temps que celui-ci. Pour cela, l'option `provides` de `Yum` permet le lister les paquets contenant cette bibliothèque.

A partir de ces deux commandes, il est possible d'écrire un script SHELL (disponible en annexe) qui recherchera, téléchargera et installera l'ensemble des paquets nécessaires à l'exécution d'un programme dynamique.

Cette technique nécessite toutefois un accès à internet pour l'utilisation du gestionnaire de paquets.

III.5.4 Contrôle du disque dans la machine virtuelle

Dans la machine hôte, le disque virtuel se manipule facilement en utilisant son chemin dans l'arborescence Windows. Mais une fois branché à la machine virtuelle, le disque est géré différemment par Linux. Cette partie expliquera la démarche et les choix qui ont été pris.

III.5.4.1 Les périphériques sous Linux

Lorsque l'on branche un périphérique, Linux va créer un fichier spécial dans le répertoire `/dev`, qui le représentera :

- les disques durs seront représentés par les fichiers `/dev/sdx` ou `/dev/hdx` (où `x` est une lettre) ;
- les clés USB par `/dev/usbmonx` ;
- le lecteur CD par `/dev/cdrom`.

Ces fichiers sont comparables aux éléments présents dans le *Poste de travail* sous Windows.

Pour accéder au contenu du périphérique, il est nécessaire de le monter dans un répertoire. Ainsi le contenu sera accessible directement dans le répertoire monté. Le procédé est comparable au branchement d'un périphérique sous Windows, qui sera accessible en tant que nouveau volume dans le *Poste de travail*, par exemple `E:` pour le lecteur CD.

Le montage d'un périphérique dans Linux s'effectue grâce à la commande `mount`, en précisant son fichier et le répertoire de montage :

```
# mount <FILE_DISK> <REP_MONT>
```

Or, on ne peut pas connaitre à l'avance quel sera le fichier, car il peut y avoir plusieurs disques de branchés, et ceux-ci ne suivent pas forcément l'ordre des numéros de port SATA (`/dev/sda` pour le port SATA 1, `/dev/sdb` pour le port 2, ...). Ainsi il est impossible de monter un disque à partir de son fichier.

III.5.4.2 UUID et périphériques

L'*UUID*, de l'anglais "Universally Unique IDentifier" ("IDentifiant Universellement Unique"), est un identifiant "unique" qui permet d'identifier les périphériques sur le système. Ce système a remplacé les noms de périphérique (`/dev/sdx`, `/dev/usbmonx`, ...) dans les fichiers

de configuration, car ces derniers peuvent changer lors du fonctionnement selon le périphérique.

L'UUID est généré aléatoirement lors de la création du périphérique. Mais il est possible de le changer en utilisant la commande `tune2fs` sous Linux :

```
# tune2fs -U <UUID> <FILE_DISK>
```

Notons que si plusieurs UUID sont identiques sur une même machine, il y aura des problèmes lors du démarrage ou de son fonctionnement.

Le montage d'un périphérique est aussi possible en utilisant l'UUID, en utilisant l'option `-U` de la commande `mount` :

```
# mount -U <UUID> <REP_MONT>
```

Il existe plusieurs façons de connaitre l'UUID d'un périphérique, comme

```
$ sudo blkid
$ ls -l /dev/disk/by-uuid
```

mais celles-ci ne donnent que la correspondance entre l'UUID et le fichier représentant le périphérique (`/deb/sdx`, ...). Or, comme on ne sait pas quel fichier représente le périphérique (expliqué en partie III.5.4.1), on ne peut pas obtenir l'UUID.

L'UUID est constant durant toute la vie du périphérique (sauf si on le change volontairement). Une solution possible serait de créer un "fichier de configuration", associant le nom du disque virtuel (sous Windows) et l'UUID du disque (sous Linux).

III.5.4.3 L'UUID VirtualBox

VirtualBox gère les différents disques virtuels en leur assignant aussi un UUID "unique". On peut le connaitre en utilisant une option de VBoxManage :

```
vboxmanage showvdiinfo <PATH_DISK_VDI>
```

Cet UUID est différent de celui que l'on peut trouver sous Linux.

Il est possible de changer l'UUID des disques virtuels en utilisant la commande :

```
vboxmanage internalcommands sethduuid <PATH_DISK_VDI> <UUID_VBOX>
```

Comme sous Linux, l'UUID doit être unique sur un même système. De plus, il est constant durant toute la vie du disque virtuel (sauf en cas de changement volontaire).

III.5.4.4 Solution

Une solution permettant de connaitre l'UUID d'un disque virtuel dans la machine virtuelle, est d'en modifier un (ou les deux) pour qu'ils soient égaux. Ainsi, pour connaitre l'UUID Linux dans la machine virtuelle, il suffira de récupérer l'UUID VirtualBox dans la machine hôte.

Le changement de l'UUID VirtualBox d'un disque virtuel pose certains problèmes. En effet, de nombreuses informations concernant les disques virtuels sont enregistrées dans des fichiers de configuration de VirtualBox ou de la machine virtuelle, dont l'UUID. Après une modification de l'UUID, il est impossible de brancher le disque virtuel à la machine virtuelle.

Ce sera donc la seconde solution qui sera utilisée, consistant à modifier l'UUID Linux pour qu'il soit identique à celui utilisé par VirtualBox.

IV Résultats - Discussion

IV.1 Manipulation des machines virtuelles

Les différentes classes développées en Python permettent la paramétrisation et le contrôle des machines virtuelles de type VirtualBox. Les fonctionnalités principalement utilisées dans la solution sont :

- Paramétrisation de la machine virtuelle : nombre de CPU, quantité de RAM ;
- Intervention sur l'état de la machine virtuelle : démarrage, mise en pause, arrêt ;
- Gestion des disques virtuels : ajout, retrait, montage et démontage dans Linux ;
- Gestion des répertoires partagés : partage, montage et démontage dans Linux ;
- Gestion de l'arborescence de Linux : création de répertoire, suppression de fichiers ou répertoires, copie ;
- Gestion des processus fonctionnant dans la machine virtuelle : pause, reprise et arrêt ;
- Échanges de fichiers ou répertoires entre la machine hôte et la machine virtuelle.

IV.2 L'interface graphique utilisateur

IV.2.1 Présentation

J'ai été libre de programmer l'interface graphique comme je le désirai, avec quelques fonctionnalités demandées. Il fallait qu'elle réponde aux objectifs du projet, c'est à dire utiliser des outils de manière simple pour l'utilisateur.

L'interface se présente de manière sobre, disposant de trois onglets.

IV.2.1.1 Interaction générale avec la machine

Il faut tout d'abord choisir une machine virtuelle parmi la liste des machines installées sur la machine hôte. Si l'utilisateur possède une machine virtuelle, utilisant Linux et paramétrée

FIGURE IV.1 – Interface graphique de la solution - Onglet 1 : l'interaction

correctement, il pourra aussi utiliser l'interface graphique pour la contrôler.

Une fois la machine choisie, les différents boutons et onglets s'activent, et l'on peut utiliser l'interface.

On peut ensuite allumer, éteindre, mettre en pause, ou redémarrer la machine virtuelle.

La machine virtuelle s'exécute en tâche de fond, c'est à dire que la fenêtre visible sur la capture d'écran I.1 n'apparait pas.

Des boutons permettent d'échanger des fichiers ou des répertoires avec la machine virtuelle. Une fenêtre de dialogue (objet de wxPython qui affiche l'arborescence de Windows) propose à l'utilisateur de choisir le fichier ou répertoire à envoyer, ou le répertoire cible de la réception. Mais il est impossible de proposer la même fenêtre pour choisir le fichier ou répertoire à recevoir, ou le répertoire cible lors de l'envoi. Il faut donc rentrer manuellement le chemin.

En cas de problème, ou de besoin particulier, l'utilisateur peut exécuter des commandes. Un thread [1] va les exécuter à l'aide de Plink (voir partie II.3 qui détail le contrôle de la machine virtuelle). L'affichage retournée par la ou les commandes est reportée "en direct" dans la fenêtre pour informer l'utilisateur.

IV.2.1.2 Les outils

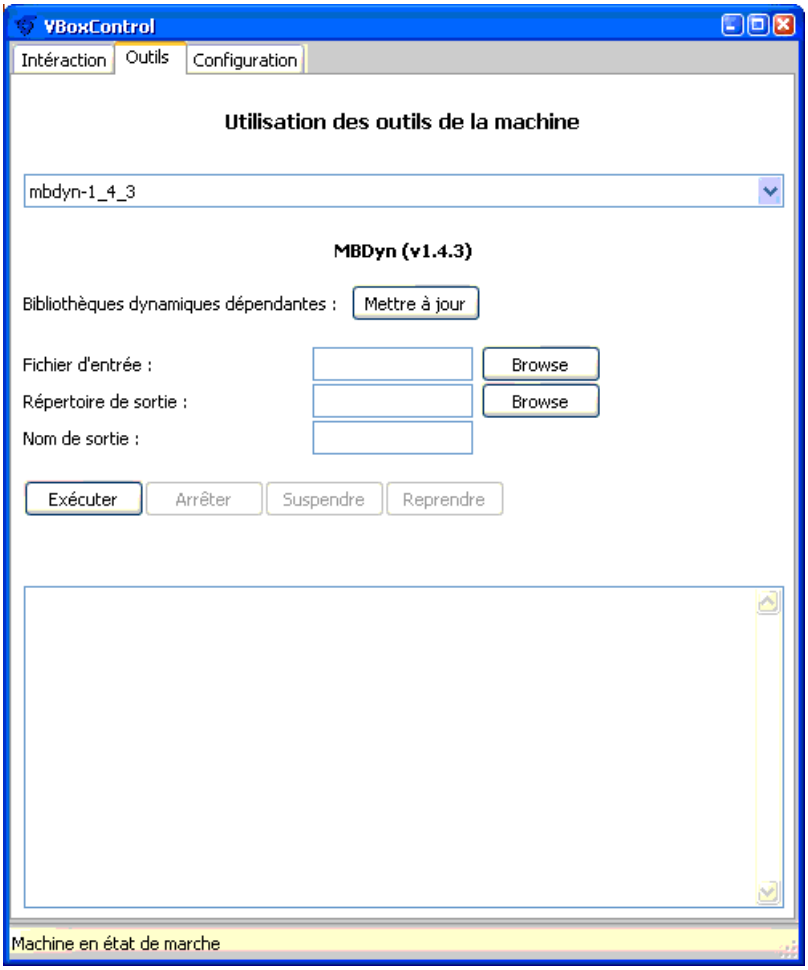

FIGURE IV.2 – Interface graphique de la solution - Onglet 2 : les outils

Chaque outil possède son interface spécifique, présente dans un fichier qui lui est dédié. L'ensemble de ces fichiers est rassemblé dans un répertoire spécifique. Lors du chargement de l'interface graphique, ces morceaux d'interface sont chargés dynamiquement.

1. Thread : tâche qui s'exécute en parallèle.

On peut ensuite choisir l'outil désiré dans la liste déroulante, qui affichera son interface juste en dessous.

Lors de l'exécution d'un outil, la démarche est généralement la même pour tous :

1. branchement du disque virtuel de l'outil sur la machine virtuelle ;
2. montage du disque, dans un répertoire spécifique ;
3. envoi des fichiers d'entrée dans la machine virtuelle ;
4. exécution du programme, en spécifiant les différentes options ;
5. réception des résultats sur la machine hôte ;
6. démontage du disque ;
7. débranchement du disque virtuel de la machine virtuelle ;

Pour régler le problème de dépendance expliqué dans la partie III.5.3, un bouton est mis à disposition de l'utilisateur pour installer les différents paquets nécessaires au fonctionnement de l'outil sélectionné.

Dans la capture d'écran IV.2, il s'agit de l'outil MBDyn qui requiert de spécifier trois options :

1. le fichier d'entrée, qui contient les différentes données qui seront traitées ;
2. le répertoire de sortie, dans lequel les résultats seront générés ;
3. le nom des fichiers générés, qui porteront une extension différente pour les différencier.

Selon l'outil l'interface peut être totalement différente, car les options ne sont pas les mêmes.

IV.2.1.3 La configuration

Cet onglet permet d'effectuer certains réglages de la machine virtuelle. Ils ne sont pas exhaustifs, et l'on pourra y ajouter d'autres options dans une version future.

Il est possible de régler le nombre de cœurs, et la quantité de mémoire vive allouée à la machine virtuelle, lorsque celle-ci est éteinte.

Plink pour l'envoi de commandes (partie II.3) et Pscp pour l'échange de fichiers (partie II.4.2) nécessitent une adresse IP, un nom d'utilisateur, ainsi qu'un mot de passe pour établir la connexion. Il est possible de modifier ces options de connexion mais il est nécessaire qu'elles soient correctes pour que cela fonctionne.

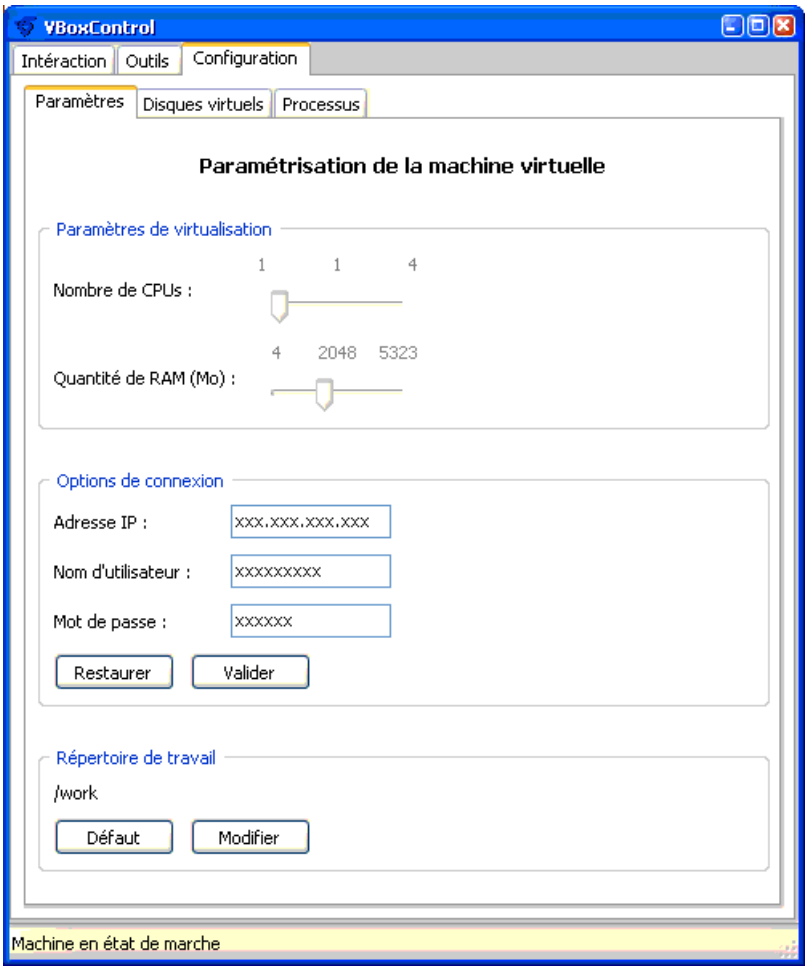

FIGURE IV.3 – Interface graphique de la solution - Onglet 3 : la configuration

Deux sous-onglets "Disques virtuels" et "Processus", permettent respectivement d'ajouter ou retirer un disque virtuel de la machine, et de tuer ou mettre en pause un processus de la machine virtuelle.

IV.2.2 Ses limites

Les différentes méthodes développées (voir partie IV.1) semblent assez lentes, car en effectuant certaines actions, l'interface se fige quelques secondes. Par exemple, pour afficher la liste des processus de la machine virtuelle dans l'onglet "Configuration", il faut environ deux secondes pour que la liste soit rafraichie.

Il serait possible de "Threader" les commandes, mais l'application serait beaucoup plus complexe à développer. Pour augmenter les performances, l'utilisation de LibVirt ou du SDK de

VirtualBox pourrait permettre des performances meilleures (détaillées en partie II.2).

Faute de temps, j'ai préféré me concentrer sur des points plus importants du projet, en utilisant des techniques moins performantes mais fonctionnelles.

IV.3 Déploiement de la solution

La solution est scindée en plusieurs parties :

- La machine virtuelle et ses outils :
  - Le système d'exploitation (Linux) est installé sur un disque virtuel assez volumineux, car il contient l'ensemble des bibliothèques dynamiques, et va grandir au fur et à mesure de son utilisation. Lors de la première utilisation, la taille est minimale car aucune bibliothèque (hormis celles installées par défaut) n'est installée. Ce disque virtuel est déployé une seule fois, car c'est la base de la solution sur laquelle viendront s'ajouter les outils ;
  - Les outils sont installés sur des disques virtuels distincts et individuels. Ainsi, lorsqu'un nouvel outil (demandé par un utilisateur) est disponible, seul son disque virtuel sera déployé sur les différents ordinateurs. L'ensemble des outils sera situé dans un répertoire dédié.
- L'interface graphique :
  - La base de l'interface graphique utilisateur devrait, comme le disque virtuel du système, être déployé une seule fois, lors de la mise en place de la solution ;
  - Les différents fichiers, contenant l'interface graphique de l'outil, seront aussi installés dans un répertoire spécifique. Lors du déploiement d'un nouvel outil, son fichier sera ajouté au répertoire pour qu'il puisse apparaitre dans l'interface graphique.

Le déploiement s'effectuera de manière optimale, facilitant aussi les futures mises à jour ou ajouts d'outils.

Conclusion

Ce stage avait pour objectif le développement d'une solution de virtualisation, permettant l'utilisation d'outils de manière simple et transparente pour l'utilisateur. De plus, la maintenance de cette solution devait être la plus efficace possible.

Ma première solution répond à toutes les attentes, en proposant une interface graphique simple d'utilisation, permettant la configuration et le contrôle de la machine virtuelle, ainsi que l'exécution des outils disponibles. Il est possible d'ajouter de nombreux outils dans la solution de virtualisation, grâce à leur installation sur des disques distincts, ce qui rendra le déploiement optimal.

La solution pourra être améliorée, notamment en ce qui concerne ses performances pour offrir le plus de fluidité à l'utilisateur. Une étude sur les besoins des utilisateurs permettrait de compléter l'éventail des fonctionnalités. Grâce aux commentaires du code source, ainsi qu'à la documentation générée, le programme pourra être maintenu très facilement.

Les principaux problèmes rencontrés concernaient la configuration de la machine virtuelle, qui requérait de nombreux réglages, notamment la configuration du réseau dans le cas d'une machine virtuelle minimale. De plus, les différents script Bash devaient être adaptés pour l'envoi avec Plink, car ils posaient de nombreux problèmes.

Cela m'a donc permis d'approfondir mes connaissances du système d'exploitation Linux, qui possède un fonctionnement bien différent de Windows, mais aussi en réseau, qui était un point clé du projet.

Webographie

[LibVirt] Red Hat, Inc.. « Libvirt 0.7.5, Application Development Guide ». 2010. http ://libvirt.org/guide/pdf/Application_Development_Guide.pdf

[Python, Doc] Python Software Foundation. « Python Documentation ». 2012. http ://python.org/doc

[Red Hat] Red Hat, Inc.. « Guide de référence ». 2005. http ://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-fr-4

[Ubuntu, Sudo] « Sudo : effectuez des tâches administratives ». 22/07/2012. http ://doc.ubuntu-fr.org/sudo

[VirtualBox] Oracle Corporation. « Oracle VM VirtualBox, User Manual ». 2012. http ://www.virtualbox.org/manual

[wxPython, Doc] wxPython Documentation. http ://wxpython.org/docs/api

Annexes

1 Configuration de la machine minimale

1.1 L'utilisateur

Lors de l'installation du système, aucun compte utilisateur n'est créé. Seul le compte administrateur est présenté, appelé *root*. Il possède tous les droits sur le système, et n'est utilisé que pour la maintenance pour des raisons de sécurité.

Pour créer un nouvel utilisateur, il existe la commande `useradd`, et il est possible de lui ajouter (ou modifier) un mot de passe à l'aide de la commande `passwd`. Cela s'effectue en mode "super-utilisateur" (su) :

```
$ su
# useradd <nom_utilisateur>
# passwd <mot_de_passe>
```

1.2 Le Grub

Le *Grub* est le programme qui permet de charger le système d'exploitation lors du démarrage de la machine. Il s'exécute après la phase de vérification du matériel (BIOS), et propose de choisir entre les différents systèmes d'exploitation installés.

Généralement les ordinateurs personnels ne possèdent qu'un système d'exploitation Windows et donc aucun menu n'est présenté. Sous Linux un menu est quand même affiché présentant les différents modes de démarrage, pour l'administration en "mode terminal" par exemple. Même si le menu n'apparait que cinq secondes environ avant de démarrer sur le système par défaut, cela ralentit le démarrage.

Il est possible d'annuler l'affichage de ce menu en modifiant cette ligne dans le fichier `/boot/grub2/grub.cfg` :

```
set timeout=5
```

En spécifiant un timeout de `0` seconde, le système démarrera directement sur le mode par défaut.

1.3 Le réseau

1.3.1 Identité de la machine

Le réseau de l'entreprise impose l'utilisation d'IP statique, et possède de plus un serveur DNS. Pour cela, il est donc nécessaire de définir l'identité de la machine sur le réseau :

- L'adresse IP
- Le masque de sous-réseau

Il est possible d'utiliser l'interface graphique pour modifier les différents paramètres, mais dans le cas de la machine virtuelle minimale cette fenêtre est indisponible. Tous les réglages doivent donc se faire en ligne de commande ou dans des fichiers de configuration.

Le fichier `/etc/sysconfig/network-scripts/ifcfg-p2p1` contient toute la configuration de l'interface réseau utilisé par la machine virtuelle. Il est mis à jour automatiquement à chaque changement d'options dans l'interface graphique.

Pour permettre la connexion au réseau et à internet, il suffira de modifier le fichier de cette manière :

```
HWADDR=08:00:27:E7:B2:0E
ONBOOT=yes
IPADDR0=10.38.21.79
NETMASK=255.255.255.0
DNS1=10.38.16.66
DNS2=10.38.16.67
\\
```

1.3.2 Le service "network"

Un *service* est un démon, c'est à dire un processus qui s'exécute en tâche de fond dans le système. Ils sont démarrés par le processus de base, *init* dont le PID 1, lors du démarrage du système.

network est le service qui gère l'ensemble du réseau sur le système, tels que la carte réseau, la table de routage, ...

Il existe plusieurs niveaux de démarrage sous Linux :

1. 0 : Il produit l'extinction du système ;

2. 1 : C'est le mode mono-utilisateur, utilisable uniquement en "root", pour la maintenance par exemple ;
3. 2, 3 : Mode multi-utilisateur ;
4. 5 : Mode multi-utilisateur, avec serveur graphique (X11) ;
5. 6 : Il produit le redémarrage du système.

Selon le niveaux utilisé, les différents services ne sont pas démarrés avec le système.

Par défaut, la machine virtuelle complète démarre en niveau 5 avec interface graphique et l'ensemble des services. Par contre, la machine minimale, qui ne dispose pas d'interface graphique, démarre en niveau 3 et ne dispose pas du service réseau.

Pour activer un service dans un niveau particulier, la technique simple et propre consiste à utiliser la commande `chkconfig`. En option, il faut préciser le niveau de démarrage, le service à modifier, puis son état de démarrage :

```
chkconfig --level <niveau> <service> <on/off>
```

Pour activer le réseau de la machine minimale, la commande à utiliser est donc :

```
chkconfig --level 3 network on
```

Une fois exécutée, le service sera automatiquement démarré lors des prochains démarrage du système.

1.3.3 La table de routage

La table de routage est une structure de données définissant le cheminement des données dans un routeur ou un ordinateur en réseau. Elle est géré dynamiquement par le système.

En effectuant les différentes configurations, j'ai remarqué que la table de routage n'était pas correcte, et que cela empêchait d'avoir accès au réseau.

Tout d'abord, une ligne de la table perturbait le bon fonctionnement. Tant qu'elle est présente, il sera impossible d'accéder au réseau. La solution consiste à supprimer cette ligne à chaque démarrage du système.

Le fichier `/etc/profile` est commun à tous les utilisateurs, et permet de configurer le Shell. Les commandes présentes dans ce fichier seront exécutées lors du démarrage. Pour effacer la ligne de la table de routage, il suffit donc d'y ajouter la commande :

```
ip route delete 10.0.0.0/8 dev p2p1
```

Ensuite, il manque certaines entrées qui permettent d'accéder à la passerelle et au proxy utilisé par l'entreprise. Il est possible de les ajouter manuellement dans le fichier

`/etc/sysconfig/network-scripts/route-p2p1`, pour que le service (network) les ajoute à chaque démarrage.

Pour cela, on ajoutera les lignes suivantes :

```
ADDRESS0=10.38.21.0
GATEWAY0=0.0.0.0
NETMASK0=255.255.255.0
ADDRESS1=0.0.0.0
GATEWAY1=10.38.21.1
NETMASK1=0.0.0.0
```

1.3.4 Le proxy

Un proxy est un intermédiaire entre différents réseaux informatiques. Il peut s'agir d'un serveur ou d'un programme. Cela permet de filtrer les données entrantes et sortantes, souvent par mesure de sécurité.

Sous Linux les paramètres du (ou des) proxy sont situés dans des variables d'environnement :

- http_proxy ;
- https_proxy ;
- ftp_proxy.

Pour que ces variables d'environnement soient définies lors du démarrage du système, il est possible d'ajouter les lignes suivantes dans le fichier `/etc/profile` ce qui les définira automatiquement :

```
export http_proxy=http://10.38.22.2:8080/
export ftp_proxy=http://10.38.22.2:8080/
export https_proxy=http://10.38.22.2:8080/
```

Pour le programme *yum*, le proxy se configure dans son fichier de configuration `/etc/yum.conf` en ajoutant les lignes suivantes :

```
proxy=http://<adresse>:<port>
proxy_username=<identifiant>
proxy_password=<mot_de_passe>
```

Dans mon cas, le proxy ne demande aucun identifiant ni mot de passe, j'ai donc ajouter la ligne :

```
proxy=http://10.38.22.2:8080
```

2 Installation d'un outil

Dans cette partie, il s'agit de l'installation de MBDyn. Tous les outils suivront cette procédure sauf pour l'étape de compilation qui sera précisée.

Dans le cas présent, on notera :

- `/dev/sdb` : le fichier représentant le disque dans la machine virtuelle ;
- `3f01e3c5-0ac1-4391-99fb-13be6595391b` : l'UUID du disque, qui est identique à celui de VirtualBox ;
- `/opt/mbdyn` : le répertoire de montage du disque ;
- `~/mbdyn-1.4.3` : le répertoire contenance la source de l'outil à installer.

On se place préalablement en mode *super-utilisateur* pour pouvoir exécuter les commandes administrateur et accéder aux répertoires protégés :

```
$ su
```

La première étape consiste à formater le disque virtuel dans un format reconnu par Linux (*ext2*) :

```
# mke2fs -F /dev/sdb
```

On change l'UUID du disque, pour avoir le même UUID que VirtualBox :

```
# tune2fs -U 3f01e3c5-0ac1-4391-99fb-13be6595391b /dev/sdb
```

On crée le répertoire de montage s'il n'existe pas, puis on monte le disque :

```
# mkdir /opt/mbdyn
# mount -U 3f01e3c5-0ac1-4391-99fb-13be6595391b /opt/mbdyn
```

Ensuite on se place dans le répertoire source de l'outil, avant la compilation et l'installation standard. C'est cette étape qui peut différer selon l'outil.

```
$ cd ~/mbdyn-1.4.3
$ ./configure --prefix=/opt/mbdyn --disable-shared --enable-static
$ make
# make install
```

L'outil est ainsi installé dans le répertoire monté, c'est à dire dans le disque virtuel. On peut démonter le disque avant de le débrancher de la machine virtuelle, en utilisant une des deux commandes :

```
# umount /opt/mbdyn
# umount /dev/sdb
```

3 Scripts SHELL

Ce script permet d'installer l'ensemble des bibliothèques dynamiques manquantes dont dépend un exécutable, en téléchargeant les paquets nécessaires.

```
#!/bin/bash

# Argument 1 : chemin de l'executable
EXE="$1"

# Tests sur l'executable d'entree
if [ ! -e "$EXE" ]
then
    echo "cible inexistante"
    exit
elif [ `file "$EXE" | grep 'executable' | wc -l` -eq 0 ]
then
    echo "cible non executable"
    exit
elif [ `file "$EXE" | grep 'dynamically linked' | wc -l` -eq 0 ]
then
    echo "executable non dynamique"
    exit
fi

# Tuer les processus utilisant YUM
kill -9 `ps aux | grep yum | cut -d " " -f 7` 2>/dev/null

# Parcours des bibliotheques manquantes
for LIB_LibNomExt in `ldd $EXE | grep 'not found' | sed "s/ => .*//"`
do
    echo -n "$LIB_LibNomExt : "
```

```
# Installee en meme temps qu'une dependance precedente
if [ `ldd "$EXE" | grep "$LIB_LibNomExt => not found" | wc -l` -eq 0 ]
then
    echo "installee"
    continue
fi

LIB_Nom=`echo "$LIB_LibNomExt" | sed "s/\.so\.[0-9]$//" | sed "s/^lib//"`

# La bibliotheque est directement dans le depot
if [ `yum search "$LIB_Nom" 2>/dev/null | grep "^$LIB_Nom.x86_64 : " | wc -l` -ne 0 ]
then
    Pack="$LIB_Nom"
else
    # Rechercher la bibliotheque dans les autres paquets
    Ligne2=`yum provides "$LIB_LibNomExt" | sed -n '2p'`
    if [ `echo $Ligne2 | grep '^No Matches found$' | wc -l` -ne 0 ]
    then
        echo "introuvable"
        continue
    else
        PACK_Nom=`echo "$Ligne2" | sed "s/ : .*$//" | sed "s/\.[a-z0-9_]*$//" \
                                 | sed "s/-[a-z0-9.]*$//" | sed "s/-[0-9.]*$//"`
        # Verifier que le paquet est dans le depot
        if [ `yum search "$PACK_Nom" | grep "^$PACK_Nom.x86_64 : " | wc -l` -eq 0 ]
        then
```

```
        echo "introuvable"
        continue
    else
        Pack="$PACK_Nom"
    fi
  fi

  # Installation du paquet
  sudo yum install -y "$Pack" >/dev/null 2>&1

  # Verification de l'installation
  if [ `ldd "$EXE" | grep "$LIB_LibNomExt => not found" | wc -l` -ne 0 ]
  then
    echo "echec de l'installation"
  else
    echo "installee"
  fi
done

# Verifications
if [ `ldd "$EXE" | grep 'not found' | wc -l` -ne 0 ]
then
  echo "Dependances manquantes"
else
  echo "Dependances installees"
fi
```

4 Résultats des tests d'échange

Ces graphiques résultent des tests d'échange développés dans la partie II.4.4.

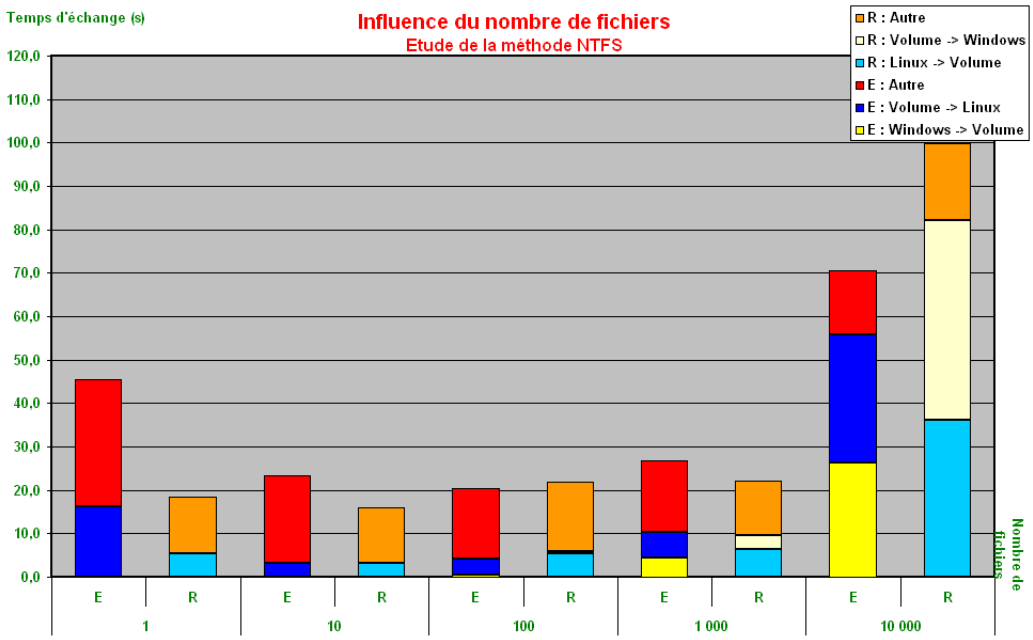

FIGURE IV.4 – Temps d'échange en fonction du nombre de fichiers
Méthode du disque virtuel

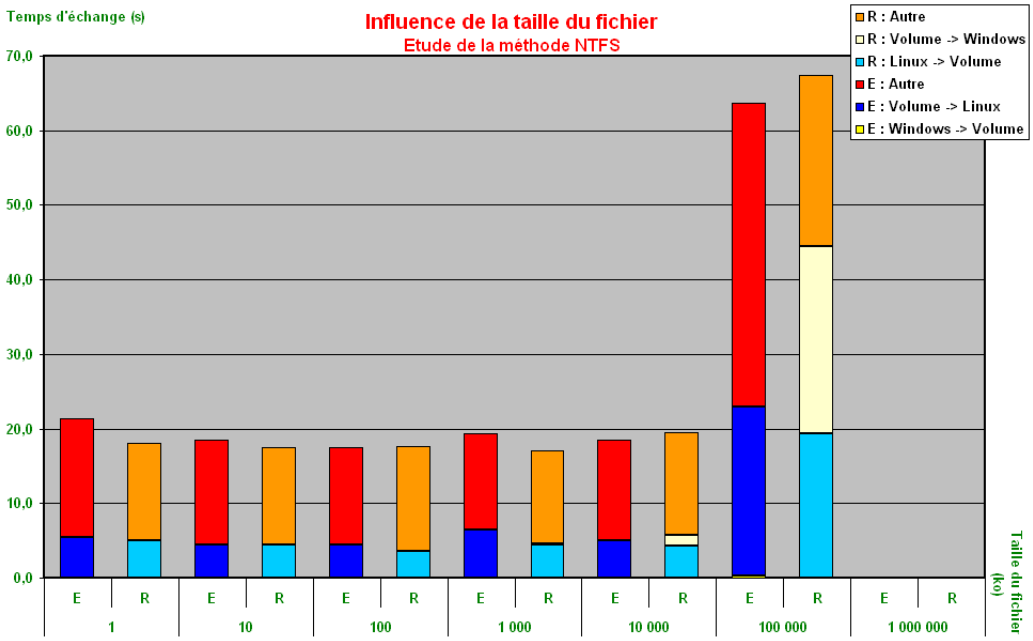

FIGURE IV.5 – Temps d'échange fonction de la taille du fichier
Méthode du disque virtuel

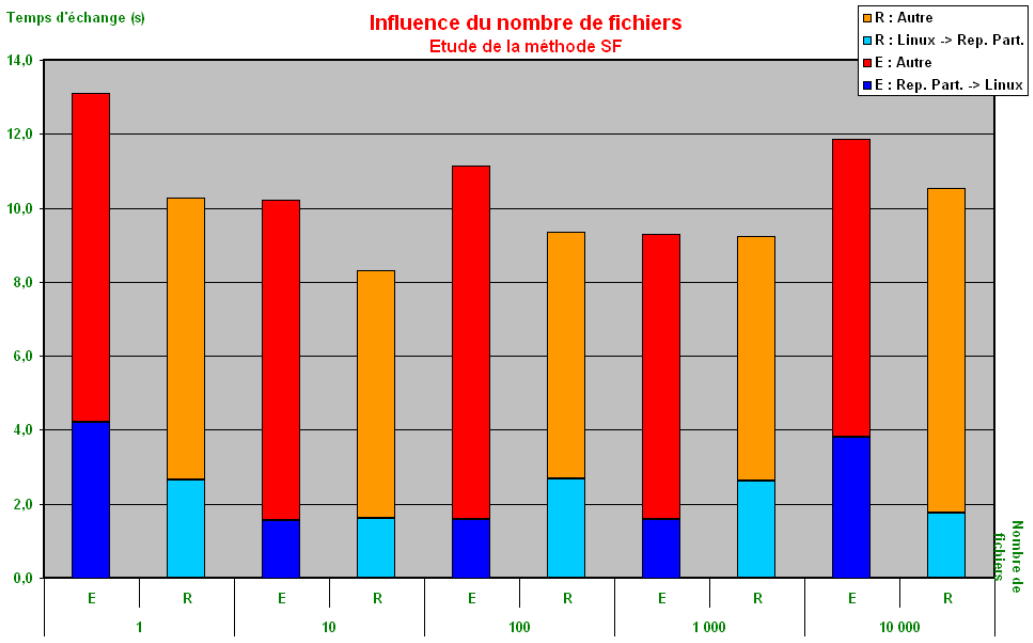

FIGURE IV.6 – Temps d'échange en fonction du nombre de fichiers
Méthode du répertoire partagé

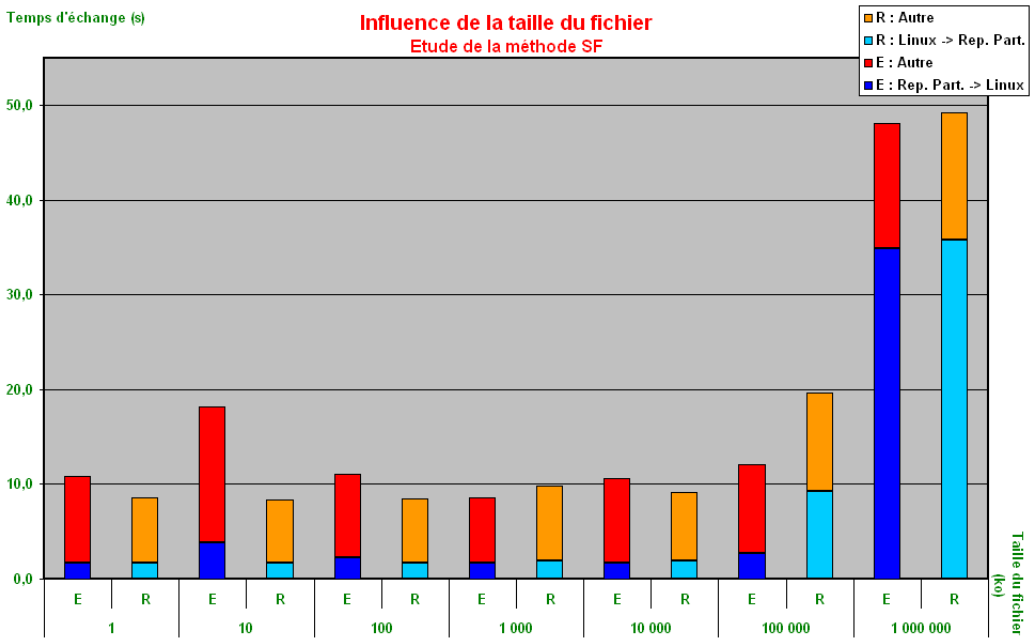

FIGURE IV.7 – Temps d'échange fonction de la taille du fichier
Méthode du répertoire partagé