Titre long

Étudiante : **Louisa Bessad,**

Encadrant : **Martin Quinson**

Table des matières

Résumé

1 Introduction

Dans le cadre de ce stage, nous allons nous intéresser aux applications distribuées. C'est-à-dire les applications dont une partie ou la totalité des ressources n'est pas stockée sur la machine où l'application s'exécute, mais sur plusieurs machines distinctes. Ces dernières communiquent entre elles via le réseau pour s'échanger les données nécessaires à l'exécution de l'application sur une plate-forme. Les applications distribuées ont de nombreux avantages ; elles permettent notamment d'augmenter la disponibilités des données en se les échangeant et en les stockant lors de communication, comme les applications Torrent (BitTorrent, Torrent...). Grâce au projet BOINC [1] par exemple, on peut également partager la puissance de calcul inutilisée de sa machine. Depuis une dizaine d'années la popularité de ces applications distribuées ne cesse de croître. Elles deviennent de plus en plus complexes avec des contraintes et des exigences de plus en plus fortes, en particulier au niveau des performances et de l'hétérogénéité des plate-formes et des ressources utilisées. Il devient donc de plus en plus difficiles de créer de telles applications mais aussi de les tester. En effet, malgré l'évolution des applications distribuées, les protocoles d'évaluation de leurs performances n'ont que peu évolués.

Actuellement, il existe trois façons de tester le comportement d'applications distribuées ; l'exécution sur plate-forme réelle, la simulation et l'émulation.

La première solution consiste à exécuter réellement l'application sur un parc de machines et d'étudier son comportement en temps-réel. Cela permet de tester l'application sur un grand nombre d'environnement. L'outil créé et développé en partie en France pour nous permettre de faire cela est **Grid'5000** [2][1], un autre outil développé à l'échelle mondiale est **PlanetLab** [3]. Néanmoins pour mettre en œuvre cette solution complexe, il faut disposer des architectures nécessaires pour effectuer les tests. Il faut également écrire une application capable de gérer toutes ces ressources disponibles. De plus, du fait du partage des différentes plate-formes entre plusieurs utilisateurs, les expériences ne sont pas forcément reproductibles.

La seconde solution consiste à faire de la simulation, c'est-à-dire à utiliser un programme appelé simulateur pour nous permettre de simuler ce que l'on souhaite étudier. Dans notre cas, pour pouvoir tester des applications distribuées sur un simulateur, on doit d'abord représenter de façon théorique l'application ainsi que l'environnement d'exécution. Pour cela, on identifie les propriétés de l'application et de son environnement puis on les transforme à l'aide de modèles mathématiques. Ainsi, on va exécuter dans le simulateur

1. https ://boinc.berkeley.edu/
2. Infrastructure de 8000 cœurs répartis dans la France entière crée en 2005. https ://www.grid5000.fr/mediawiki/index.php/Grid5000 :Home
3. Crée en 2002, cette infrastructure de test compte aujourd'hui 1340 noeuds. http ://www.planet-lab.org

le modèle de l'application dans un environnement également modélisé et non l'application réelle. Cette solution est donc facilement reproductible, simple à mettre en œuvre du moment que nous savons modéliser notre application et permet de prédire l'évolution du système étudié grâce à l'utilisation de modèles mathématiques. De nos jours, les simulateurs, tel que **SIMGRID**[2, 3], peuvent simuler des applications distribuées mettant à contribution des milliers de noeuds. Néanmoins, avec la simulation on ne peut valider qu'un modèle et pas l'application elle même puisqu'on exécute le modèle de l'application dans le simulateur.

La troisième solution consiste à faire de l'émulation, cela signifie que nous allons exécuter réellement l'application mais dans un environnement virtualisé grâce à un logiciel. On fera ainsi croire à l'application qu'elle s'exécute sur une machine autre que l'hôte. Cette solution représente un intermédiaire entre la simulation et l'exécution sur plate-forme réelle. En effet les actions de l'application sont réellement exécutées sur la machine hôte mais on lui fait croire grâce au simulateur qu'elle se trouve dans un environnement différent de la machine hôte. De plus, cette émulation peut-être faite *off-line* ; on sauvegarde les actions de l'application sur disque et on les rejoue plus tard dans le simulateur ou *on-line* ; les actions sont directement reportées dans le simulateur et on bloque l'application durant le temps nécessaire calculé par le simulateur.

Dans le cadre du projet Simterpose c'est l'émulation qui a été choisie pour tester des applications distribuées. En effet la simulation n'était pas une bonne solution puisque nous voulons valider les applications et non leur modèle. En ce qui concerne l'exécution sur plate-forme réelle il y avait trop de contraintes matérielles à satisfaire. Il existe deux types d'émulation pour les applications distribuées ; la virtualisation standard et la "légère". On parle de virtualisation "légère" quand on souhaite tester des applications sur une centaine d'instances. Dans ce rapport nous allons présenter en section 2 les méthodes utilisées pour faire de la virtualisation légère : limitation et interception. Puis en section 3 nous verrons les projets qui existent aujourd'hui pour ce type de virtualisation. Pour finir en section 4 nous expliquerons pourquoi dans le cadre du projet Simterpose c'est la virtualisation légère par interception qui a été choisie et comment elle fonctionne.

2 Méthodes possibles pour la virtualisation légère

Il existe actuellement deux méthodes permettant de faire de la virtualisation légère. La première est une émulation par limitation ou dégradation également appelée virtualisation standard et la seconde est une émulation par interception.

2.1 Virtualisation standard

Avec cette première méthode on place la couche d'émulation au-dessus de la plateforme réelle (comme un hyperviseur pour une VM). De fait, la puissance de l'émulateur dépend de la puissance de la machine hôte et ne peux pas dépasser les capacités de cette dernière. En effet, des machines plus puissantes que l'hôte répondrait plus rapidement que ce dernier à une demande d'une application. Or le délai de réponse géré par l'émulateur ne peut-être inférieur à celui de l'hôte sinon l'hôte n'aurait pas le temps de faire les calculs demandés et de répondre à l'émulateur qui répondrait à l'application. De plus, en choisissant de placer l'émulation comme une sur-couche cela permet de limiter l'accès aux ressources pour les applications. En effet les applications ne pourront pas passer la couche d'émulation pour accéder aux ressources localisées sur la machine hôte. Les requêtes des applications distribuées seront arrêtées par l'émulateur. C'est lui qui s'occupera de récupérer les ressources demandées par les applications. Il existe différent outils permettant de faire de mettre en place cette virtualisation, on trouve notamment **cgroup**, **netstat** et **cpuburner**. Cette solution à l'avantage d'être simple à mettre en œuvre puisque l'on se basse sur la machine hôte. Néanmoins elle est assez contraignante du fait qu'on ne puisse pas émuler des architectures plus performantes que l'hôte. De plus à écrire deux derniers points négatifs à éclaircir.

2.2 Émulation par interception

Dans le cas de l'émulation par interception, pour faire croire à l'application qu'elle s'exécute sur une machine autre que l'hôte on va utiliser deux outils ; un simulateur pour virtualiser l'environnement d'exécution et une API qui va attraper toutes les communications de l'application avec l'hôte et qui les transmettra ensuite au simulateur. Les calculs de l'application seront effectués sur la machine hôte mais c'est le simulateur qui calculera le temps de réponse à l'application. Pour cela il fera un rapport entre le temps d'exécution du calcul sur la machine hôte (fourni par l'API), la puissance de l'hôte et celle des machines de l'environnement que l'on simule. Le temps de l'application sera donc celui du simulateur et non le temps réel. En effet, l'application quand elle fait un calcul pense être sur une autre machine avec des performances différentes, elle est donc capable de savoir combien de temps prends une certain calcul sur son architecture. Hors sur l'hôte ce calcul ne prendra pas le même temps et l'application se retrouvera avec un temps prévu et un temps qui ne correspondent pas ce qui est problématique. Avec cette solution on ne se contente pas de faire de l'interception d'action et du rejeu par l'émulateur comme c'est le cas avec l'émulation par limitation. On va intercepter les actions des applications et faire

de la médiation, autrement dit on va modifier les actions avant de les laisser s'exécuter sous le contrôle de l'émulateur.

Une application distribuée peut vouloir communiquer avec l'hôte soit pour effectuer de simples calculs (SEB), soit pour effectuer des requêtes de connexion ou de communication avec d'autres applications sur le réseau. Quand l'émulateur intercepte une communication venant d'un des processus d'une application, il modifie les caractéristiques de cette dernière pour qu'elle puisse s'exécuter sur la machine hôte. Quand la machine hôte renvoie une réponse à l'application, elle est également interceptée par l'émulateur pour que l'application ne voit pas le changement d'architecture. En même temps, il envoie au simulateur des données concernant le temps d'exécution de l'action sur la machine hôte pour qu'il puisse calculer le temps d'exécution sur la machine simulée. Les délais calculés par le simulateur sont soit des temps de calculs soit des temps de connexion. *Quand le simulateur a terminé le calcul du temps de réponse nécessaire il l'envoie à l'émulateur qui l'envoie à l'application en plus du résultat afin de mettre à jour l'horloge de l'application.* Ainsi les calculs sont réellement exécuter sur la machine, les communications réellement émises sur le réseau géré par le simulateur et c'est le temps de réponse fourni par le simulateur qui va influencer l'horloge de l'application permettant ainsi d'imiter un environnement distribué. Finalement les applications ne communiquent plus directement entre elles puisque toute communication est interceptée par l'émulateur, puis gérée par le simulateur qui s'occupe du réseau.

Mettre deux schémas de action interceptes, test modifie, renvoie, attrape réponse, simulateur, retour application, un quand simple calcul l'autre quand connexion

Pour intercepter ces actions, il faut d'abord choisir à quel niveau se place l'interception : code source ou binaire. Mais il faut également choisir sur quel type de symbole utilisée par l'application pour exécuter ses actions se fera la médiation qui suit l'interception et avec quel outil. En effet une application peut communiquer avec le noyau via différentes abstractions. Elle peut soit utiliser les fonctions d'interaction directe avec le noyau que sont les appels systèmes, soit utiliser les différentes abstractions fournies par le système d'exploitation : bibliothèques (fonctions systèmes de la libc par exemple) ou les fonctions POSIX dans le cas d'un système UNIX.

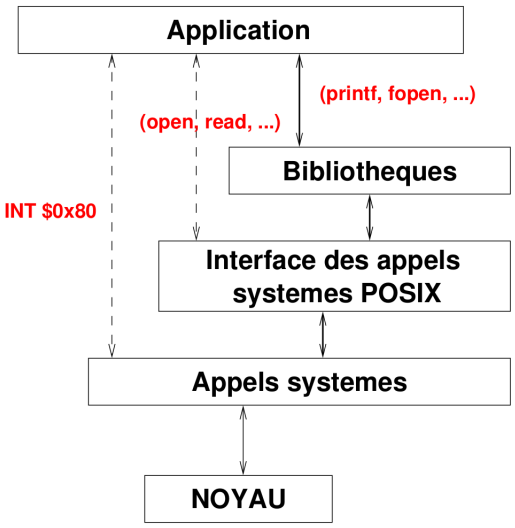

Figure 1 – Communications possibles entre le noyau et une application

Nous allons donc voir comment on peut faire de l'interception sur un fichier source puis sur un binaire, puis nous verrons comment faire de la médiation sur différents symboles : appels de fonctions et appels systèmes.

2.2.1 Action sur le fichier source

2.2.2 Action sur le binaire

2.2.3 Médiation directe des appels de fonctions

linker : LD_PRELOAD

linker got injection

2.2.4 Médiation des Appels Système

ptrace

uprobe

seqcomp/bpf

3 État de l'art

3.1 CWRAP

3.2 RR

3.3 Distem

3.4 MicroGrid

3.5 DETER

3.6 ROBOT

4 Symbôles sur lesquels faire de la médiation

4.1 Organisation générale

4.2 Les communications réseaux

4.3 Les thread

4.4 Le temps

4.5 DNS

5 Conclusion

Références

[1] Raphaël Bolze, Franck Cappello, Eddy Caron, Michel Daydé, Frédéric Desprez, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Julien Leduc, Noredine Melab, et al. Grid'5000 : a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, 2006.

[2] Henri Casanova. Simgrid : A toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 430–437. IEEE, 2001.

[3] Martin Quinson. SimGrid : a Generic Framework for Large-Scale Distributed Experiments. In *9th International conference on Peer-to-peer computing - IEEE P2P 2009*, Seattle, United States, September 2009. IEEE.