École Polytechnique
Promotion X2011
Chloé MACUR

RAPPORT DE STAGE DE RECHERCHE

Émulation d'applications distribuées sur des plates-formes virtuelles simulées

. . . . . .

NON CONFIDENTIEL

Option :	Informatique
Champ de l'option :	Conception des Systèmes Informatiques
Directeur de l'option :	Olivier Bournez
Directeur de stage :	Martin Quinson
Dates du stage :	31 mars - 10 août 2014
Nom et adresse de l'organisme :	LORIA 615 Rue du Jardin Botanique 54506 Vandœuvre-lès-Nancy

Résumé

La complexité croissante des environnements distribués nécessite de disposer d'outils afin de tester les applications distribuées. Dans le projet SimTerpose nous proposons d'utiliser la technique d'émulation, qui consiste à exécuter des applications réelles dans un environnement virtuel. SimTerpose intercepte pour celà les actions des différentes applications pour leur présenter un environnement virtuel qui est simulé par le simulateur SimGrid. Afin de modifier les calculs et les communications des applications, nous utilisons l'outil *ptrace* qui permet l'interception et le remaniement des appels système.

Abstract

The increasing complexity of distributed systems calls for tools that test distributed applications. In SimTerpose project we use an emulation approach: we run real applications in virtual environnements. To that extend, SimTerpose intercepts the actions of the applications and offers them a virtual environment simulated by the SimGrid simulator. To modify computations and communications between the applications we intercept and transform system calls with *ptrace*.

Table des matières

Remerciements

Je tiens à remercier mon tuteur de stage Martin Quinson pour m'avoir accueillie dans son équipe et encadrée durant ce stage de recherche. Je remercie plus généralement les membres de l'équipe AlGorille pour leur patience et le temps qu'ils m'ont consacrés.

Introduction

Les systèmes distribués tels que les grilles, les systèmes pairs-à-pairs ou encore le cloud sont de plus en plus courants. En effet, ils offrent de grandes capacités de calcul et permettent de partager et décentraliser des données. De nombreuses applications s'exécutent donc sur de tels systèmes. Toutefois le développement de ces applications ainsi que l'évaluation de leurs performances ou de leur résistance aux pannes sont extrêmement complexes. Il est ainsi nécessaire de disposer d'outils de tests adaptés aux applications distribuées.

Nous présentons ici le projet SimTerpose qui permet d'exécuter et de tester des applications distribuées réelles dans un environnement virtuel. Pour ce faire SimTerpose intercepte les actions des applications réelles, comme les calculs et les communications, et modifie les réponses à ces actions. Des délais peuvent par exemple être ajoutés pour modifier l'environnement perçu par les applications. L'environnement virtuel est alors simulé à l'aide du simulateur SimGrid [1], développé par l'équipe AlGorille [2] en collaboration avec d'autres équipes.

Nous présentons dans une première partie les motivations et les objectifs de notre projet, avant de décrire la méthodologie retenue. Nous détaillons par la suite notre contribution au projet SimTerpose. Enfin nous passons en revue les travaux qui se rapprochent du nôtre avant de conclure.

1. Voir `http://simgrid.gforge.inria.fr/`.
2. Voir `http://www.loria.fr/la-recherche/equipes/algorille/`.

I Contexte

1 Motivations

Si les systèmes distribués offrent une grande puissance de calcul et une redondance de l'information, ils sont toutefois plus complexes que les systèmes centralisés. L'évaluation des performances ou de la résistance aux pannes des applications distribuées est rendue difficile par la nature distribuée et parfois hétérogène de l'environnement. Il est donc crucial de disposer de tests qui soient adaptés et faciles d'utilisation.

Il existe différentes méthodes pour tester des applications distribuées. La première consiste à exécuter les applications réelles sur une plate-forme distribuée réelle comme Grid'5000 [2]. Cependant, en plus de la complexité inhérente à la manipulation de ce type d'environnement, il faut pouvoir disposer de l'infrastructure adéquate au moment de l'expérimentation. Il est par ailleurs difficile de reproduire une expérience dans des conditions strictement identiques car les plateformes sont souvent partagées avec d'autres utilisateurs, dont les actions influencent les conditions expérimentales.

Inversement, dans l'approche par simulation on modélise à la fois les applications et l'environnement, tandis que les interactions entre les deux sont calculées à l'aide d'un simulateur. Puisque les applications réelles ne sont pas utilisées directement, il est nécessaire de les réécrire avec l'interface du simulateur afin de les modéliser.

Enfin, l'émulation consiste à exécuter les applications réelles dans un environnement virtuel. Un exemple serait d'ajouter une couche d'émulation par dessus un cluster entier afin d'obtenir l'environnement virtuel désiré. Toutefois, cette méthode requiert là encore l'utilisation d'une plate-forme réelle complexe et est donc relativement compliquée à mettre en place. Notre objectif consiste à créer un outil simple, à la manière d'une machine virtuelle, permettant d'exécuter des applications distribuées réelles sur des plates-formes qui n'existent pas. Nous n'utilisons pas la virtualisation complète au niveau matériel, cet objectif est réalisé par d'autres moyens : nous choisissons d'utiliser une approche par émulation fondée sur l'interception des actions de l'application à étudier, en utilisant le simulateur

SimGrid pour simuler l'environnement virtuel. Cette démarche offre l'avantage de ne pas nécessiter d'infrastructure lourde, et permet également de tester une application arbitraire sans disposer de son code source.

2 SimGrid

SimGrid [1] est un ensemble d'outils de simulation qui permet l'étude d'applications distribuées dans des environnements hétérogènes. Ce projet, développé par l'équipe AlGorille en collaboration avec d'autres équipes depuis plus de dix ans, a pour but de faciliter la recherche sur les systèmes parallèles et distribués à large échelle tels que les grilles, les systèmes pair à pair, les plates-formes de calcul hautes performances ou les clouds. Toutefois, il est impossible d'utiliser des applications réelles avec le simulateur SimGrid : il faut réécrire les applications à l'aide d'une des interfaces de SimGrid. SimTerpose vise ainsi à fournir un moyen d'utiliser SimGrid avec des applications réelles.

3 Objectifs

Notre projet se doit d'être simple d'utilisation et facilement déployable tant sur un ordinateur personnel que sur un cluster. Il est donc nécessaire que notre outil puisse exécuter plusieurs instances d'une application sur un même système et autorise la personnalisation de la plate-forme virtuelle, qu'il s'agisse des caractéristiques des nœuds ou de la topologie du réseau. SimTerpose doit également pouvoir générer les traces d'une application pour la rejouer ensuite dans le simulateur SimGrid, nous autorisant ainsi à tester la performance et la robustesse de façon parfaitement reproductible. L'étude de ces traces facilitera grandement l'analyse du comportement de l'application distribuée. La résistance aux pannes des applications pourra également être évaluée en modifiant à loisir l'environnement — variation de la bande passante ou de la latence — ainsi qu'en injectant des fautes : rupture de liens ou arrêt de machine. Par ailleurs nous souhaitons que SimTerpose permette de tester les applications sans avoir accès à leur code source et donc *a fortiori* sans les modifier.

Nous nous proposons donc de tester les applications réelles sur une plate-forme virtuelle simulée par le simulateur SimGrid.

FIGURE I.1 – Les applications semblent s'exécuter sur une plate-forme distribuée, alors qu'elles sont en réalité sur une seule machine.

II Démarche employée

Nous signalons ici que notre travail fait suite aux études de M. Guthmuller [3] et G. Serrière [10] lors de stages précédents dans l'équipe AlGorille. Nos résultats s'appuient donc fortement sur leurs travaux.

1 Émulation par interception

Suite aux travaux de M. Guthmuller [3] nous choisissons ici d'utiliser la méthode de l'*émulation par interception* afin de pouvoir tester des applications réelles dans un environnement distribué virtuel. En effet, dans d'autres travaux l'environnement virtuel est souvent obtenu en ajoutant une couche d'émulation à une plate-forme réelle. On peut ainsi réduire les capacités de l'environnement hôte en ajoutant des délais à chaque calcul et communication. Cette technique d'*émulation par dégradation* notamment utilisée par l'émulateur Distem [1] [9] ne permet toutefois pas d'émuler une plate-forme plus puissante que l'originale.

Dans SimTerpose, nous avons donc fait le choix d'intercepter les actions des différentes applications et de les rediriger vers un simulateur. Les calculs sont véritablement exécutés sur la plate-forme réelle, leur durée d'exécution est mesurée puis réinjectée dans le simulateur. Les communications sont récupérées puis modifiées afin de remanier l'environnement perçu par l'application. La réponse de l'environnement à ces actions, comme par exemple les délais correspondant aux temps de calcul et de communication, est alors calculée par un simulateur. Nous avons utilisé le simulateur SimGrid puisqu'il est au cœur du projet de l'équipe AlGorille. Outre ses performances, SimGrid est un outil ayant fait ses preuves en termes de validité de ses modèles et de stabilité de ses interfaces dans le temps.

1. Voir `http://distem.gforge.inria.fr/`.

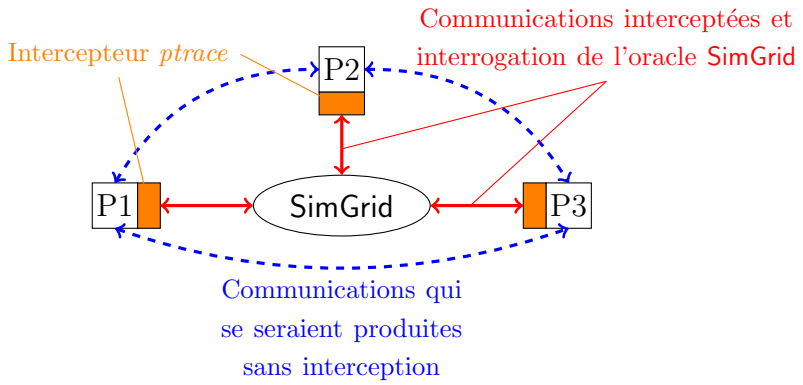

FIGURE II.1 – Principe général de l'émulation par interception. Trois processus (P1, P2, P3) sont exécutés, et leurs actions sont interceptées, puis redirigées vers le simulateur SimGrid.

2 Utilisation de *ptrace*

Différentes méthodes d'interception ont été envisagées par M. Guthmuller avec pour critères la capacité d'interception, le coût en terme de performances et la facilité d'utilisation. L'outil *ptrace*[2] a ainsi été retenu afin d'intercepter les appels systèmes des applications et de modifier les registres de ces appels. *ptrace* est un appel système qui permet à un processus de contrôler l'exécution d'un autre processus. L'utilisateur peut par exemple insérer des points d'arrêt pour le *debugging*, puisque le processus tracé s'arrête à chaque fois qu'un signal lui est délivré. Le contrôleur est alors prévenu et peut modifier le processus suivi pendant son arrêt puis continuer son exécution.

Nous choisissons ici d'utiliser les appels système comme points d'arrêt, comme `bind`, `connect` ou `recv`. Pour chaque appel, l'observateur sera appelé deux fois. Lors de l'entrée dans l'appel, les registres contiennent le numéro de l'appel système désiré ainsi que les arguments nécessaires. Une modification des registres à ce moment permet ainsi de substituer des arguments comme le destinataire d'un `send`, ou même de modifier le type de l'appel système qui sera effectué (voire de neutraliser complètement l'appel système). Le processus père sera également appelé à la sortie de l'appel système et peut à nouveau modifier les registres, comme la valeur de retour. L'outil *ptrace* permet également de lire ou d'écrire directement dans la mémoire du processus tracé via les arguments `PEEK_DATA` et `POKE_DATA`. On notera que *ptrace* nécessite de prendre en compte les différentes

2. Voir `http://man7.org/linux/man-pages/man2/ptrace.2.html`.

architectures matérielles qui ont des conventions d'appels système distinctes. Pour cette raison, nous avons choisi de limiter l'usage de SimTerpose à des architectures AMD 64 bits.

3 Médiation

L'application réelle à étudier voit donc ses actions redirigées vers le simulateur, qui calcule la réaction de la plate-forme virtuelle. Selon les caractéristiques de cet hôte simulé, les phases de calcul et de communication peuvent ainsi être retardées. On peut réellement modifier la perception des applications en interceptant les fonctions liées au temps, afin par exemple de simuler une exécution plus rapide que sur la plate-forme réelle.

Le travail de G. Serrière [10] établit deux modes de fonctionnement pour les communications dans SimTerpose : la médiation totale et la traduction d'adresses. Puisque nous exécutons plusieurs applications localement, il est nécessaire de leur présenter un environnement apparemment distribué. SimTerpose doit donc faire le lien entre les adresses et les ports du réseau simulé et ceux du réseau local.

3.1 Traduction d'adresse

Les communications dans les réseaux se font via des *sockets* (interfaces de connexion). Les sockets renferment les informations concernant les ports et les adresses des correspondants, et interviennent dans les appels système régissant les communications.

Dans le cas de la traduction d'adresse nous établissons une table de traduction entre les adresses réelles locales et les adresses globales simulées. Lors de l'établissement de connexion nous modifions les arguments des appels systèmes comme `bind`, `connect` ou `send` afin de réellement utiliser le réseau local. Au retour des appels systèmes, nous rétablissons les arguments initiaux afin que les applications voient un environnement distribué. Nous gardons en mémoire ces tables de correspondances afin de les utiliser lors de phases de communication (`sendto`, `recvfrom`). Dans ce mode d'utilisation, nous laissons donc réellement le noyau gérer les communications, seuls les interlocuteurs sont altérés.

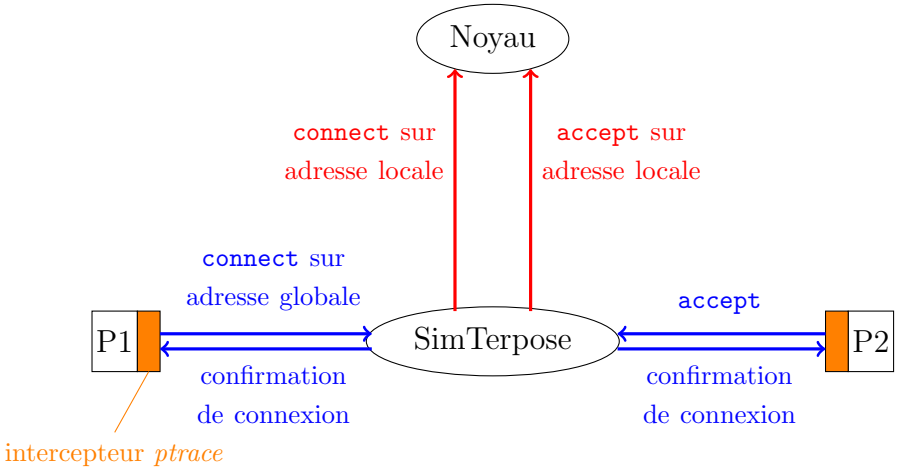

FIGURE II.2 – Principe de la traduction d'adresses. Un processus cherche à se connecter sur le réseau global. SimTerpose traduit le couple adresse-port et modifie l'appel système en conséquence.

3.2 Médiation totale

Inversement, la médiation totale ne laisse pas réellement les applications établir de connexion, ni communiquer à l'aide de socket. Dans ce mode, nous utilisons *ptrace* avec l'argument `POKE_DATA` afin d'écrire directement dans la mémoire du processus. Nous pouvons donc conserver les adresses simulées et tous les appels systèmes visant à établir une connexion sont neutralisés par SimTerpose.

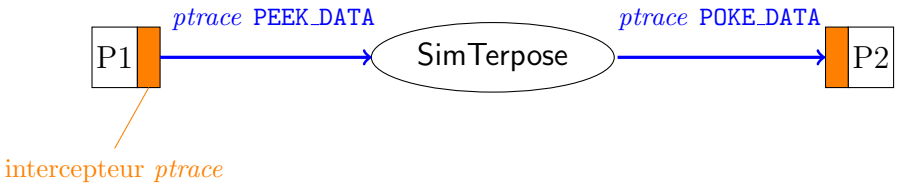

FIGURE II.3 – Principe de la médiation totale dans le cas d'une communication. Nous allons directement chercher les arguments dans la mémoire du processus émetteur, et nous les insérons dans la mémoire du destinataire.

III Outil opérationnel et ajout de fonctionnalités

1 Mise en conformité et correction du code existant

Le prototype sur lequel repose notre travail date d'il y a deux ans. Puisque la partie simulation est assurée par SimGrid nous nous sommes dans un premier temps familiarisés avec ce simulateur. Nous avons ensuite entrepris d'en utiliser la version la plus récente et avons donc actualisé la manière dont SimGrid était appelé.

Par ailleurs le code que nous avons repris était une démonstration de faisabilité, qui présentait un certain nombre de fonctionnalités implémentées. Toutefois, ce projet n'était pas fonctionnel puisqu'il ne permettait pas d'effectuer une simulation. Nous avons donc dans un premier temps entrepris de produire un code conforme aux bonnes pratiques de génie logiciel et aux conventions du projet SimGrid (unification des conventions d'appel, réorganisation de la structure du projet, écriture de tests). Cette réingénierie fut une étape fondamentale afin de parvenir à rendre SimTerpose opérationnel.

2 Retour sur l'outil d'interception

Lors des travaux de M. Guthmuller comparant différentes méthodes d'interception, l'outil Uprobes [4] n'avait pas pu être utilisé car il était encore au stade du développement. Nous avons ainsi étudié la possibilité de remplacer *ptrace* par Uprobes. En effet, cet outil permet d'éviter la multiplication des changements de contexte qui nuit aux performances. Uprobes permet de pénétrer dynamiquement dans une application et de collecter des informations de débogage et de performances sans interruption. Il est possible de placer des points d'interception à n'importe quelle adresse de code, en spécifiant un gestionnaire à invoquer lorsque le

point d'arrêt est atteint. La fonction d'enregistrement `register_uprobe()` indique quel processus est à sonder, où le point d'arrêt doit être inséré et quel gestionnaire doit être appelé lorsque le point d'arrêt est atteint. Uprobes fonctionne ainsi : lorsqu'un point d'arrêt est enregistré, l'instruction sondée est copiée, l'application arrêtée, le début de l'instruction sondée est remplacé par la routine à invoquer, puis l'exécution de l'application reprend. Uprobes autorise également l'utilisateur à mettre un point d'arrêt au retour d'une fonction, et permet de gérer à la fois les fonctions dans l'espace utilisateur et dans le noyau. Cet outil extrêmement prometteur nécessite toutefois la création de modules noyau si l'on désire personnaliser le gestionnaire à invoquer. L'avantage offert par cette approche ne semble ainsi pas suffisant au vu de la complexité imposée aux utilisateurs. Nous maintenons donc l'utilisation de *ptrace*, qui nous permet d'éviter cette complication.

3 Modification de l'interface de programmation utilisée

SimGrid offre plusieurs interfaces de programmation (*Application Programming Interface*, API). SimDAG permet l'étude d'applications structurées comme un graphe orienté acyclique (*Directed Acyclic Graph*, DAG). On utilise SMPI pour simuler des applications *Message Passing Interface* (MPI) en interceptant les primitives MPI. Enfin MSG est destiné à l'étude des applications *Concurrent Sequential Processes* (CSP) et permet ainsi d'analyser des algorithmes d'ordonnancement par exemple sur des grilles.

SimTerpose utilisait initialement SimDAG, pour lequel les tâches de communications sont relativement complexes : un envoi de message nécessite de créer trois tâches (une tâche d'envoi, une tâche de transfert et une tâche de réception) ainsi que des dépendances entre ces tâches afin de respecter la chronologie : la réception ne peut avoir lieu avant l'envoi. En raison de la complexité d'utilisation de SimDAG dans notre projet et de l'évolution de l'interface MSG nous avons fait le choix de changer complètement d'API.

Le mode de fonctionnement global de la version utilisant SimDAG est le suivant : un gestionnaire unique traite tous les processus actifs. Les appels système sont donc interceptés indifféremment du processus appelant. Cette méthode nous oblige à utiliser une machine à états centralisée pour gérer les processus qui sont en état d'attente, à cause d'un appel bloquant par exemple.

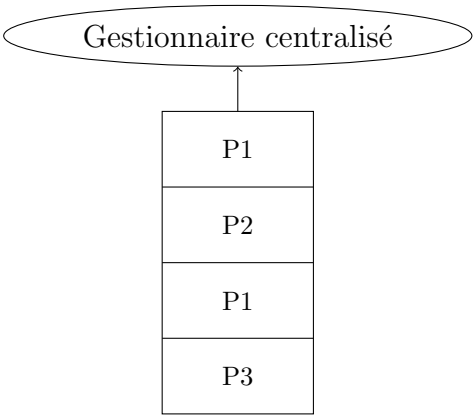

FIGURE III.1 – Gestion des processus à l'aide de SimDAG

Inversement dans la version qui utilise MSG, nous créons un processus MSG par processus qui s'exécute réellement. Il existe donc un gestionnaire par processus : en cas d'appel bloquant les autres processus continuent à s'exécuter normalement ce qui facilite la gestion des processus. On a alors un *thread* du simulateur pour chaque processus applicatif étudié.

Cette approche n'était pas réalisable en 2012 car les threads portant le code utilisateur dans le simulateur devaient s'exécuter en exclusion mutuelle. S'il était possible d'écrire des prototypes d'applications spécifiquement pour ce mécanisme, des deadlocks applicatifs pouvaient apparaître avec des applications réelles.

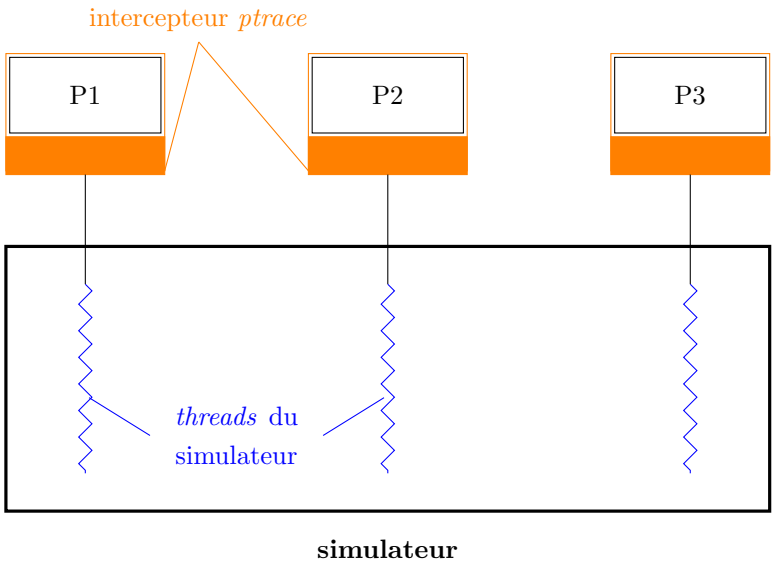

FIGURE III.2 – Gestion parallélisée des processus à l'aide de MSG : chaque *thread* du simulateur joue réellement les actions de l'application correspondante

4 Appels liés au temps

Nous souhaitons que l'écoulement du temps perçu par l'application soit complètement virtuel. Pour ce faire nous devons intercepter toutes les fonctions liées au temps et les modifier afin de renvoyer aux applications le temps désiré. En raison du grand nombre de fonctions différentes, il paraît judicieux d'intercepter directement les appels système `time`, `clock_gettime` et `gettimeofday`.

Toutefois en rectifiant ces appels système à l'aide de *ptrace* nous n'obtenons pas les résultats escomptés. Cela s'explique par l'existence de la bibliothèque *Virtual Dynamic Shared Object* (VDSO). Cette bibliothèque a pour but de réduire les coûts dûs aux changements de contexte. En effet, un appel système impose deux changements de contexte ce qui coûte plusieurs centaines de cycles processeur. VDSO permet alors de retrouver l'heure (qui se trouve dans le contexte noyau) sans quitter le contexte utilisateur.

Au lieu d'effectuer systématiquement l'appel système, la bibliothèque conserve ainsi la valeur d'un appel récent et interpole la valeur demandée. Une solution envisageable est donc de désactiver le VDSO en passant les options *vdso=0* et *vsyscall=native* au démarrage du noyau. Nous avons testé cette approche qui est parfaitement fonctionnelle. Cependant, cette solution réduit les performances de

l'application, en plus de forcer l'utilisateur à redémarrer sa machine pour modifier ces paramètres.

L'alternative choisie est donc d'utiliser le préchargement de bibliothèques via `LD_PRELOAD`. Au lieu d'intercepter les appels système nous nous plaçons au niveau des appels de bibliothèque. Nous allons ainsi créer une bibliothèque qui surcharge toutes les fonctions liées au temps et ce seront nos propres fonctions qui seront exécutées au lieu de celles d'origine. La partie délicate consiste à ne pas oublier de fonctions, bien plus nombreuses que les appels système.

5 État de l'art

Nous nous sommes intéressés à un projet proche du nôtre : cwrap [1], dont le but est de tester des applications réseaux. Ce projet utilise le préchargement de bibliothèque via l'éditeur de liens dynamique. La bibliothèque préchargée contient des fonctions qui viennent écraser les fonctions désirées ce qui permet de surcharger sélectivement les fonctions d'autres bibliothèques partagées. Dans le cas de cwrap les bibliothèques surchargent notamment tous les appels qui servent à communiquer à l'aide de sockets, afin de router les communications sur le réseau local. cwrap n'agit pas uniquement sur les sockets, mais également les résolutions de noms de domaines (*Domain Name System*, DNS). Il permet de plus de simuler des droits utilisateur (`setuid`).

Notre choix d'utiliser *ptrace* nous autorise à réaliser l'interception au niveau des appels systèmes. L'interception au niveau des bibliothèques est plus fastidieuse car il est nécessaire de surcharger toutes les fonctions sous peine d'être contourné par l'application. L'alliance de ces deux méthodes nous semble donc être la solution optimale.

Le projet MicroGrid [11] se rapproche également de notre travail. Le réseau est simulé et s'appuie sur un ensemble de machines physiques. Les actions de l'application sont à nouveau interceptées au niveau des bibliothèques via `LD_PRELOAD`. Bien qu'il n'existe plus de version maintenue de MicroGrid, cette approche a été réutilisée dans d'autres projets. TimeKeeper [5] permet à chaque *linux container* (LXC) d'avoir sa propre horloge virtuelle, et offre la possibilité de faire des pauses ou des sauts dans le temps. Il s'agit d'un module noyau qui a été intégré au framework CORE (*Common Open Research Emulator*, un émulateur de réseaux en temps réel qui émule les hôtes par virtualisation et simule les liens). La fonction *gettimeofday* est réimplémentée pour renvoyer un temps virtuel, si toutefois cette option est activée. Un autre projet, *Integrated simulation and emulation using*

1. Voir `http://cwrap.org/`.

adaptive time dilation [6], a pour but d'allier les avantages de la simulation et de l'émulation. Toutefois, lorsque le simulateur est surchargé il peut prendre du retard sur le temps réel et donc introduire un délai lorsqu'il envoie des informations à l'émulateur. Ce projet consiste donc à dilater le temps afin de garder la synchronisation. Le facteur de dilatation est adapté en fonction de la charge du CPU de l'hôte physique. La méthode utilisée est la virtualisation avec l'hyperviseur KVM et le remplacement de *gettimeofday* par une fonction *get_virtual_time*.

Conclusion

Suite du travail

A l'heure où nous écrivons ce rapport notre projet n'est pas encore terminé puisque le stage continue jusqu'au 10 août. Nous allons dans un premier temps finaliser l'implémentation de SimTerpose à l'aide de l'API MSG. Nous ajouterons ensuite des fonctionnalités manquantes, en combinant *ptrace* avec des interceptions `LD_PRELOAD` s'inspirant de cwrap afin de traiter les fonctions liées au temps, aux permissions (`setuid`) ou encore à la médiation des noms de domaines (DNS).

Nous projetons par la suite d'évaluer les performances de notre projet. Nous souhaitons en particulier estimer la taille maximale des expériences que l'on peut simuler. A titre de comparaison, les expériences par virtualisation complète peinent à excéder la centaines de processus simulés par machine physique [8]. Notre virtualisation ultra légère nous permet d'espérer au moins quelques dizaines de milliers de processsus dans les mêmes conditions, puisque SimGrid a déjà été utilisé avec succès pour simuler plusieurs millions de processus repliés sur une machine unique [7]. Nous envisageons également de quantifier le réalisme de notre simulation, en comparant une expérience réelle avec celle exécutée avec SimTerpose.

Par ailleurs, puisque SimTerpose propose d'utiliser deux modes de fonctionnement — médiation totale ou traduction d'adresse — nous nous proposons de confronter l'efficacité et la précision de ces deux méthodes. Nous nous attendons à ce que la traduction d'adresse soit plus efficace, étant attendu que les `PEEK_DATA` et `POKE_DATA` sont relativement coûteux en temps.

Perspectives

Nous avons présenté un outil simple d'utilisation et facile à déployer, permettant d'exécuter et de tester des applications distribuées réelles sans disposer de leur code source. Les actions des applications sont interceptées et modifiées pour être exécutées dans un environnement virtuel simulé par le simulateur SimGrid. Nous proposons d'allier les méthodes d'interception au niveau des appels système

— via *ptrace* — et au niveau des bibliothèques de fonctions — via `LD_PRELOAD` — afin de couvrir tous les types d'action : calculs, communications, fonctions liées au temps.

Nous pourrons à l'aide de SimTerpose modifier l'environnement virtuel en injectant diverses fautes dans la simulation. Notre travail facilitera ainsi l'analyse des applications distribuées en testant leur performance et leur robustesse.

Bibliographie

[1] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid : a generic framework for large-scale distributed experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, 2008.

[2] Franck Cappello et al. Grid'5000 : a large scale, reconfigurable, controlable and monitorable Grid platform. In *6th IEEE/ACM International Workshop on Grid Computing - GRID 2005*, Seattle, USA, États-Unis, November 2005.

[3] Marion Guthmuller, Lucas Nussbaum, and Martin Quinson. Émulation d'applications distribuées sur des plates-formes virtuelles simulées. In *Rencontres francophones du Parallélisme (RenPar'20)*, Saint Malo, France, May 2011. RR-7536 RR-7536.

[4] J. Keniston, A. Mavinakayanahalli, P. Panchamukhi, and V. Prasad. Ptrace, utrace, uprobes : Lightweight, dynamic tracing of user apps. In *Proceedings of the 2007 Linux Symposium*, pages 215–224, 2007.

[5] Jereme Lamps, David M. Nicol, and Matthew Caesar. Timekeeper : A lightweight virtual time system for linux. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '14, pages 179–186, New York, NY, USA, 2014. ACM.

[6] Hee Won Lee, David Thuente, and Mihail L. Sichitiu. Integrated simulation and emulation using adaptive time dilation. In *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '14, pages 167–178, New York, NY, USA, 2014. ACM.

[7] Martin Quinson, Cristian Rosa, and Christophe Thiery. Parallel Simulation of Peer-to-Peer Systems. In *CCGrid 2012 – The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 668–675, Ottawa, Canada, May 2012. IEEE. RR-7653 RR-7653.

[8] Benjamin Quétier, Vincent Neri, and Franck Cappello. Scalability comparison of four host virtualization tools. *Journal of Grid Computing*, 5(1) :83–98, 2007.

[9] Luc Sarzyniec, Tomasz Buchert, Emmanuel Jeanvoine, and Lucas Nussbaum. Design and Evaluation of a Virtual Experimental Environment for Distributed Systems. In *PDP2013 - 21st Euromicro International Conference on*

Parallel, Distributed and Network-Based Processing, pages 172 – 179, Belfast, Royaume-Uni, February 2013. IEEE. RR-8046 RR-8046.

[10] Guillaume Serrière. Simulation of distributed application with usage of syscalls interception. `http://webloria.loria.fr/~quinson/Research/Students/2012-master-simterpose-rapport.pdf`.

[11] H. J. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien. The microgrid : A scientific tool for modeling computational grids. *Sci. Program.*, 8(3) :127–141, August 2000.