Pré-rapport

Louisa Bessad

Abstract

Ce papier est un résumé conçernant le CGRA DART et les mécanismes de détection/correction d'erreurs qu'il implémente. Nous verrons notamment comment les concepteurs de DART ont pu améliorer de façon significative ses performances grâce à un nouvelle technique de détection de type *on-line error-detecting* appelée *residue code modulo 3*.

1 Introduction

mettre sur 2 pages une présentation de l'entreprise et des activités, et sur 10 à 20 pages la présentation du contexte de la mission, les problématiques, et les solutions envisagés. Plus ce qu'on a déjà fait si on a déjà attaqué le vrai travail.

1. entreprise activité (relier au 3 maybe)
2. pour les tester 3 méthodes: l'exécution réelle, la simulation de l'exécution en environnement modélise et l'émulation cas où les applications s'exécutent réellement mais sur des environnements virtuels reprendre sujet stage à développer dans 1
3. Dans le cadre de ce stage nous allons nous intéresser aux applications ditribués à large échelle et comment on peut les tester et évaluer leurs perfs via une combinaison d'émulation et de simulation en utilisant SIMGRID et Simterpose.
4. SIMGRID et Simteprose = projet europeen blablabla
5. nous ce sera la partie émulation sur laquelle nous travaillerons
6. plan maybe 2.pourquoi faire de l'émulation simulation et pas les deux autres 3. comment ça marche réellement 4.ce qu'il y a à faire et pour quand

2 Pourquoi choisir l'émulation simulée

Il existe trois façons de tester des applications distribuées. La première consiste à exécuter réellement l'application sur un parc de machines et d'étudier le comportement de l'application en temps-réel, ce que fait actuellement **Grid'5000** mettre citation. Néanmoins pour mettre en œuvre cette solution complexe il faut disposer des architectures nécessaires pour effectuer les tests. De part le partage des différentes plateformes entre divers utilisateurs les expériences ne sont pas

forcément reproductibles. Une deuxième solution consiste à simuler l'exécution des applications à l'aide d'un simulateur tel que **SIMGRID** mettre citation. On doit alors modéliser l'application ainsi que l'environnements d'exécution grâce à des modèles mathématiques. Le problème avec la simulation est que l'on exécute pas vraiment l'application, on ne peut alors valider qu'un modèle et pas l'application puisqu'on réécrit l'application selon le modèle. La troisième solution consiste à faire de l'émulation, c'est-à-dire que nous allons exécuter réellement l'applications mais dans en environnement virtuels. Ainsi nous aurons un simulateurs qui gérera l'environnement, l'application qui s'exécutera réellement sur la machine hôte et une API qui fera le lien entre l'application qui s'exécute et l'environnement simulé. On fera ainsi croire à l'application qu'elle s'exécute sur une machine autre que l'hôte. Il existe deux façons de faire de l'émulation: la dégradation et l'interception. Dans la première on rajoute la couche d'émulation au-dessus de la plateforme réelle (comme un hyperviseur pour une VM). Mais cela nous empêche d'émuler des machines plus puissante que l'hôte car le délai de réponse géré par l'émulateur ne peut-être inférieur à celui de l'hôte sinon l'hôte n'a pas le temps de faire les calculs nécessaires à l'application. Cette solution choisie par **Distem**mettre citation est donc limitée à la capacité des plateformes à notre disposition. Dans le cas de l'interception pour faire croire à l'application qu'elle s'exécute sur une machine autre que l'hôte on va attraper toutes ses communications via une API du simulateur qui ensuite les transmetra au simulateur. Ce dernier s'occupera de calculer le temps de réponses de ces commumnications en se basant sur les performances de la machine qu'on est en train de simuler. Les calculs seront effectués sur la machine hôte mais le temps de réponse à l'application sera géré par le simulateur en fonction de l'architecture que l'on simule en utilisant le temps d'exécution de la machine hôte injectés dans le simulateur et en comparant les performances des deux architectures. Ainsi le temps de l'application sera celui du simulateur et non le temps réel. Cette solution est implémentée dans **Simterpose**mettre citation. Dans notre cas d'émulation simulation, nous allons utiliser SIMGRID comme simulateur et Simterpose comme API de ce simulateur. Nous aurons donc Simterpose qui permettra d'utiliser SIMGRID avec des applications réelles en leur faisant croire qu'elles s'exécutent sur des machines distribuées. Maintenant que nous savons comment nous allons tester les applications distribuées nous allons voir comment fonctionne notre "émulateur" Simterpose.

3 Comment fonctionne Simterpose

3.1 Interception des actions

Simterpose est l'API qui permet d'intercepter les communications de l'application avec la machine sur laquelle elle s'exécute. Sans cela l'application se rendrait compte que l'environnement réel ne correspond pas à celui dans lequel elle pense être. Une applications distribuées peut vouloir communiquer avec l'hôte soit pour effectuer des calculs (SEB), soit pour communiquer avec d'autres applications sur le réseau. Quand Simterpose intercepte une communication venant de l'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, Simterpose l'intercepte également et la modifie 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 calculer le temps sur la machine simulé puis envoie ce temps à l'application en plus du résultat pour mettre à jour son horloge. Ainsi les calculs sont réellement exécuter sur la machine et le temps de réponse fournir par le simulateur qui va influencer l'horloge de l'application permet d'imiter un environnement distribué.

MEttre un schéma de action interceptes, test modifie, renvoie, attrape reponse, simulateur, retour application

Les délais calculés par le simulateur sont soit des temps de calculs soit des temps de connexion. Les actions éxécutées par l'application peuvent-être soit de simples calculs soit des requêtes de connexion ou communication réseau.

Pour intercepter ces actions, il faut d'abord choisir ce qu'on intercepte exactement et avec quel outil. En effet une application peut communiquer avec une machine via différentes abstractions.

schéma marion??? liste... tout seul tout seul puis le lien entre les deux

Nous allons intercepter les actions que sont les AS fait par l'applciation ainsi nous sommes sûr d'intercepter tous les types de communications que l'applicatione est susceptible d'initier avec l'OS. Il existe de nombreux outils en matières d'interception d'AS. On trouve notamment a remplir dire pourquoi abandonné. Pour intercepter les éventuels AS d'une applciation nous allons donc utiliser **ptrace**citation. Cet outil qui est lui-même un AS va permettre à un processus d'en contrôler un autre. Ainsi un processus va dire qu'il souhaite être contrôle "espionnée" et un autre processus le contrôlera "processus espion". Ce dernier va spécifier sur quels évènements du processus "espionné" il veut être notifié, dans notre cas ce sera les AS que l'on considérera comme point d'arrêt. *Cet AS permet L'avantage de l'AS ptrace c'est qu'il permet d'écrire et de lire directement dans la mémoire des processus.* Ainsi quand l'application voudra faire un AS quelconque elle sera bloquée et l'AS ptrace sera lancé et notifiera le "processus espion", ce dernier fera les modifications nécessaires dans les registres de l'applications pour conserver la virtualisation de l'environnement puis il rendra la main à l'application pour que l'AS ait lieu. Au retour de l'AS le "processus espionné" sera de noveau stopper et un ptrace sera envoye à l"espion" qui remodifiera les informations nécessaires et rendra la main à l'application qui sortira de son AS avec un résultat exécuté sur la machine hôte et un temps d'exécution et une horloge fournie par le simulateur. Néanmoins il a été montré dans un précédent stage que l'AS ptrace est inefficace voir inutile en ce qui concerne tous les AS temporel que l'application voudrait faire. *Les AS "time", "clock_gettime", "gettimeofday" avec **ptrace** ne sont pas possibles, d'où l'alliance avec LD_PRELOAD* Par exemple lors d'un gettimeofday l'AS n'est pas lancé on répond directement au niveau de la bibliothèque ainsi on n'arrive même pas au niveau de l'AS, donc ptrace ne fait rien.

On pourrait alors se dire qu'une bonne solution serait d'intercepter les actions de l'application au niveau des bibliothèques pour cela il existe l'éditeur de lien dynamique **LD_PRELOAD** rajouter des trucs sur VDSO. Ainsi on va créer notre propre bibliothèque de fonction pour chaque fonction que l'application serait susceptible d'utiliser. Ainsi dans notre nouvelle fonction on ajoutera toutes les modifications nécessaires pour maintenir notre environnement simulé puis on fera appel à la vraie fonction celle de base et on préchargera cette bibliothèque avant les autres grâce à LD_PRELOAD. Ainsi nos fonctions passeront avant les fonctions de base de l'OS. Le problème de cette solution est que si l'application fait un appel à une fonction que l'on n'a pas réécrit rien ne

l'empchera de faire l'AS directement sans avoir modifier ses paramètres, de fait cette solution n'est pas complète. Ainsi on peut voir que LD_PRELOAD résout les problèmes de ptrace et inversement, une solution choisie lors d'un précédent stage est donc d'allier les deux. Maintenant que nous savons ce que nous devons intercepter et comment l'intercepter nous allons voir ce que nous devons modifier pour pouvoir maintenit notre émulation simulation.

Pour faire face à ce problème il a été choisi d'utiliser l'éditeur de lien dynamique ***LD_PRELOAD****, ce dernier intercepte les appels de l'application au niveau des bibliothèques. Pour cela on va créer une bibliothèque.*

3.2 Modification à effectuer

References