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

Marion Guthmuller, Lucas Nussbaum, Martin Quinson

LORIA - Équipe AlGorille

Nancy - Université

17 février 2011

Plan

Introduction (1/2)

Contexte : évaluation des applications distribuées

- **Exécution sur plate-forme réelle** (PlanetLab, Grid'5000)
  - ☺ Exécution directe de l'application étudiée
  - ☹ Mise en œuvre lourde et reproduction difficile
- **Simulation** : mise en œuvre d'un modèle de l'application
  - ☺ Rapide et facile, reproductibilité parfaite
  - ☹ Interface spécifique au simulateur, application réelle à reprogrammer

Introduction (2/2)

- **Émulation** : substitution de l'environnement par un logiciel
  - *Émulation par dégradation* : réduction des caractéristiques de la plate-forme réelle (Modelnet, DieCast, Emulab, Wrekavoc)
    ☹ Nécessite le déploiement d'une infrastructure complexe, impossible d'émuler une plate-forme plus puissante
  - *Émulation par simulation* : modification de la perception de l'application
    - MicroGrid (dernière version en 2004) → inadapté aux systèmes actuels
    - **Notre projet : Simterpose**

Introduction (2/2)

- **Émulation** : substitution de l'environnement par un logiciel
  - *Émulation par dégradation* : réduction des caractéristiques de la plate-forme réelle (Modelnet, DieCast, Emulab, Wrekavoc)
    ☹ Nécessite le déploiement d'une infrastructure complexe, impossible d'émuler une plate-forme plus puissante
  - *Émulation par simulation* : modification de la perception de l'application
    - MicroGrid (dernière version en 2004) → inadapté aux systèmes actuels
    - **Notre projet : Simterpose**

Introduction (2/2)

- **Émulation** : substitution de l'environnement par un logiciel
  - *Émulation par dégradation* : réduction des caractéristiques de la plate-forme réelle (Modelnet, DieCast, Emulab, Wrekavoc)
    ☹ Nécessite le déploiement d'une infrastructure complexe, impossible d'émuler une plate-forme plus puissante
  - *Émulation par simulation* : modification de la perception de l'application
    - MicroGrid (dernière version en 2004) → inadapté aux systèmes actuels
    - **Notre projet : Simterpose**

Plan

Objectifs et critères de succès

Un émulateur simple :

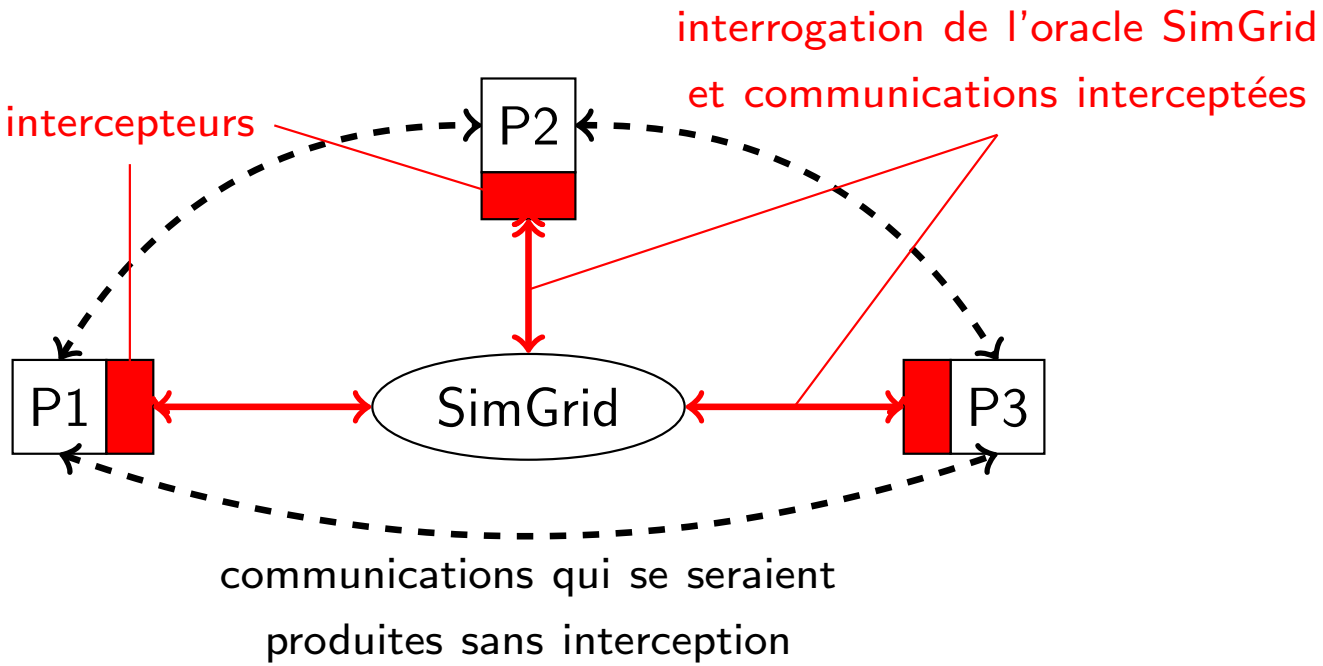
- Ne dépend pas d'une infrastructure complexe
- Déployable sur un ordinateur portable ou un cluster
- Permettant le **repliement de processus** sous un **large éventail de conditions** et **l'étude du comportement** de l'application **pendant son exécution**

Critères de succès :

- **Facilité d'utilisation** : pas besoin d'accès *root* ou de recompilation du noyau
- **Performance** : exécution d'un grand nombre de noeuds virtuels
- **Précision** : comportement virtuel aussi proche du comportement réel
- **Généricité** : différentes plates-formes, différents langages
- **Stabilité et maintenance** : utilisation de composants déjà stables

Plan

Principe général



Solution d'émulation basée sur le simulateur SimGrid

Défis

1. Sélection des actions à intercepter
   - Actions liées à la création et à l'identité des processus
   - Actions liées aux entrées/sorties et aux communications
   - Actions liées au temps : travail en temps réel ou virtualisation du temps
2. Reproduire l'impact de la plate-forme virtuelle sur l'exécution
   - Calcul virtuel du temps d'exécution (l'opération n'est pas réalisée réellement)
   - Mesure de la durée de chaque action (l'opération est réellement réalisée) et calcul de la différence par rapport à la plate-forme virtuelle
3. Interception des actions de l'application : 3 approches
   - Interception par virtualisation complète (exécution sur machines virtuelles)
   - Interception au niveau du middleware utilisé par l'application
   - Interception au travers d'outils systèmes (`ptrace`, Valgrind)

Défis

1. Sélection des actions à intercepter
   - Actions liées à la création et à l'identité des processus
   - Actions liées aux entrées/sorties et aux communications
   - Actions liées au temps : travail en temps réel ou virtualisation du temps
2. Reproduire l'impact de la plate-forme virtuelle sur l'exécution
   - Calcul virtuel du temps d'exécution (l'opération n'est pas réalisée réellement)
   - Mesure de la durée de chaque action (l'opération est réellement réalisée) et calcul de la différence par rapport à la plate-forme virtuelle
3. Interception des actions de l'application : 3 approches
   - Interception par virtualisation complète (exécution sur machines virtuelles)
   - Interception au niveau du middleware utilisé par l'application
   - Interception au travers d'outils systèmes (`ptrace`, Valgrind)

Défis

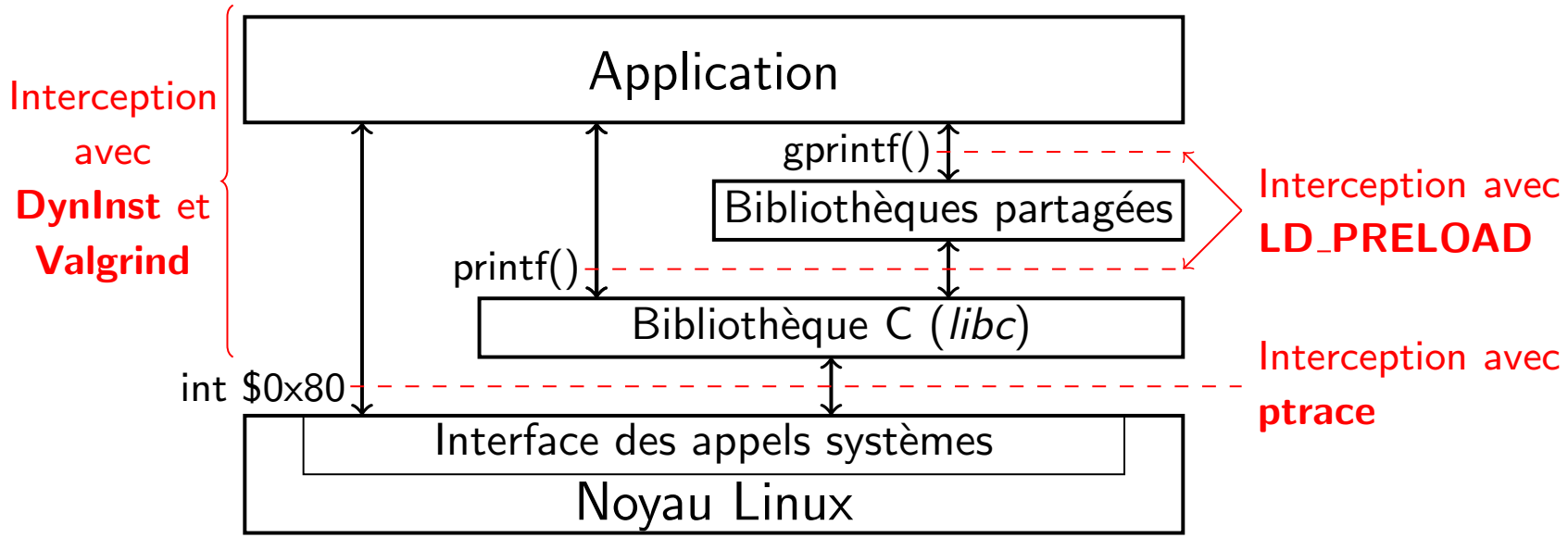
1. Sélection des actions à intercepter
   - Actions liées à la création et à l'identité des processus
   - Actions liées aux entrées/sorties et aux communications
   - Actions liées au temps : travail en temps réel ou virtualisation du temps
2. Reproduire l'impact de la plate-forme virtuelle sur l'exécution
   - Calcul virtuel du temps d'exécution (l'opération n'est pas réalisée réellement)
   - Mesure de la durée de chaque action (l'opération est réellement réalisée) et calcul de la différence par rapport à la plate-forme virtuelle
3. Interception des actions de l'application : 3 approches
   - Interception par virtualisation complète (exécution sur machines virtuelles)
   - Interception au niveau du middleware utilisé par l'application
   - Interception au travers d'outils systèmes (`ptrace`, Valgrind)

Plan

Évaluation de méthodes d'interception

- Critères :
  1. Capacité d'interception (d'après son niveau dans la pile logicielle)
  2. Coût et impact sur les performances de l'application
  3. Facilité d'utilisation

- 4 approches :



Valgrind et DynInst

Valgrind : outil de programmation pour le profilage de code

- Déroute les appels aux fonctions à intercepter vers d'autres fonctions
- Travail au niveau du code binaire de l'application cible
- ☹ Phase de décompilation/recompilation pour réaliser l'interception
  $\Rightarrow$ Temps d'exécution multiplié par 7.5

DynInst : API permettant l'insertion de code pendant l'exécution

- Au niveau de la couche application
- ☺ Surcoût faible sur le temps d'exécution
- ☹ API complexe, dépendance logicielle

Valgrind et DynInst

Valgrind : outil de programmation pour le profilage de code

- Déroute les appels aux fonctions à intercepter vers d'autres fonctions
- Travail au niveau du code binaire de l'application cible
- ☹ Phase de décompilation/recompilation pour réaliser l'interception
  $\Rightarrow$ Temps d'exécution multiplié par 7.5

DynInst : API permettant l'insertion de code pendant l'exécution

- Au niveau de la couche application
- ☺ Surcoût faible sur le temps d'exécution
- ☹ API complexe, dépendance logicielle

LD_PRELOAD et ptrace

LD_PRELOAD : préchargement d'une bibliothèque d'interception

- Modification du comportement de l'application de façon indirecte
- Au niveau des appels de bibliothèques
- ☺ Faible coût et facilité d'utilisation
- ☹ Surcharge les fonctions des bibliothèques mais pas des appels systèmes

ptrace : contrôle de l'exécution d'un autre processus

- Interception au niveau du noyau
- ☹ Interface pas normalisée POSIX → problème de portabilité
- ☺ Coût moyen, interception bas niveau très efficace
- Alternative : **Uprobes** (en cours de développement)

LD_PRELOAD et ptrace

LD_PRELOAD : préchargement d'une bibliothèque d'interception

- Modification du comportement de l'application de façon indirecte
- Au niveau des appels de bibliothèques
- ☺ Faible coût et facilité d'utilisation
- ☹ Surcharge les fonctions des bibliothèques mais pas des appels systèmes

ptrace : contrôle de l'exécution d'un autre processus

- Interception au niveau du noyau
- ☹ Interface pas normalisée POSIX → problème de portabilité
- ☺ Coût moyen, interception bas niveau très efficace
- Alternative : **Uprobes** (en cours de développement)

Tableau récapitulatif

	Valgrind	**DynInst**	**LD_PRELOAD**	**Ptrace**
Niveau d'interception	application	application	bibliothèques	noyau
Capacité d'interception	☹ moyenne	☹ moyenne	☹ faible	☺ très bonne
Coût	☹ important	☺ assez faible	☺ faible	☹ moyen
Facilité d'utilisation	☹ complexe	☹ très complexe	☺ simple	☹ assez complexe

Plan

Implémentation d'un prototype avec ptrace

1. Principe similaire à l'utilitaire `strace`
   - Extraction des appels systèmes et des paramètres pour chaque processus
   - Identification des processus communiquants
2. Détermination des périodes de calcul
3. Validation sur une application pair-à-pair réelle : BitTorrent

Exemple de trace client/serveur

```
      Timestamp  pidX  wall_time  cpu_time  diff_wall  diff_cpu type  syscall local_addr:port  remote_addr:port
pidY return  param

23:15:18:938060  6976      19234     12000          0     12000      (v)fork
6977
23:15:18:944354  6976      25537     16000       6303      4000      (v)fork
6978
23:15:21:957648  6976    3038838     16000    3013301         0      (v)fork
6989
23:15:21:969823  6977    3031988         0          0         0   in    recv 127.0.0.1: 2226  127.0.0.1:34024
6989
23:15:21:970159  6989      12697         0          0         0   in    send 127.0.0.1:34024  127.0.0.1: 2226
6977
23:15:21:970356  6989      12895         0        198         0  out    send 127.0.0.1:34024  127.0.0.1: 2226
6977    512  (4, "..." , 512)
23:15:21:970471  6977    3032640         0        652         0  out    recv 127.0.0.1: 2226  127.0.0.1:34024
6989    512  (5, "..." , 512)
23:15:21:970594  6989      13133         0        238         0   in    recv 127.0.0.1:34024  127.0.0.1: 2226
6977
23:15:21:970791  6977    3032963         0        323         0   in    send 127.0.0.1: 2226  127.0.0.1:34024
6989
23:15:21:970966  6977    3033136         0        173         0  out    send 127.0.0.1: 2226  127.0.0.1:34024
6989    512  (5, "..." , 512)
23:15:21:971104  6989      13643         0        510         0  out    recv 127.0.0.1:34024  127.0.0.1: 2226
6977    512  (4, "..." , 512)
```

Plan

Conclusion

- Conception d'un émulateur pour permettre l'évaluation d'applications distribuées sur des plates-formes virtuelles simulées
  1. Interception des actions (calculs, communications) de l'application
  2. Calcul de la réponse de la plate-forme à ces actions par le simulateur SimGrid
- Évaluation de 4 méthodes d'interception :
  - Valgrind
  - DynInst
  - `LD_PRELOAD`
  - `ptrace`
- Implémentation d'un prototype basé sur l'approche `ptrace`
- Validation sur une application pair-à-pair réelle : BitTorrent

Perspectives

- Émulation *offline* : permettre le rejeu des traces acquises sur simulateur
- Émulation *online* : interfacer l'intercepteur d'actions avec le simulateur
  - Mise en place d'un lanceur
  - Démarrage automatique des nœuds sur la plate-forme simulée
- Émulation *distribuée* : distribution des processus sur différentes machines
  - Utilisation d'un cluster
  - *Framework* intégré pour l'étude d'applications réelles sur plates-formes simulées