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

Marion Guthmuller, Lucas Nussbaum et Martin Quinson

LORIA / Nancy-Université – Équipe AlGorille

RenPar'20 (Rencontres francophones du Parallélisme) - 13 Mai 2011

Plan

Contexte : évaluation des applications distribuées

3 méthodes :

- **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
- **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
    - **Simterpose**

Contexte : évaluation des applications distribuées

3 méthodes :

- **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
- **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
    - **Simterpose**

Contexte : évaluation des applications distribuées

3 méthodes :

- **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
- **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
    - **Simterpose**

Contexte : évaluation des applications distribuées

3 méthodes :

- **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
- **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
    - **Simterpose**

Objectifs et critères de succès

Un émulateur simple :

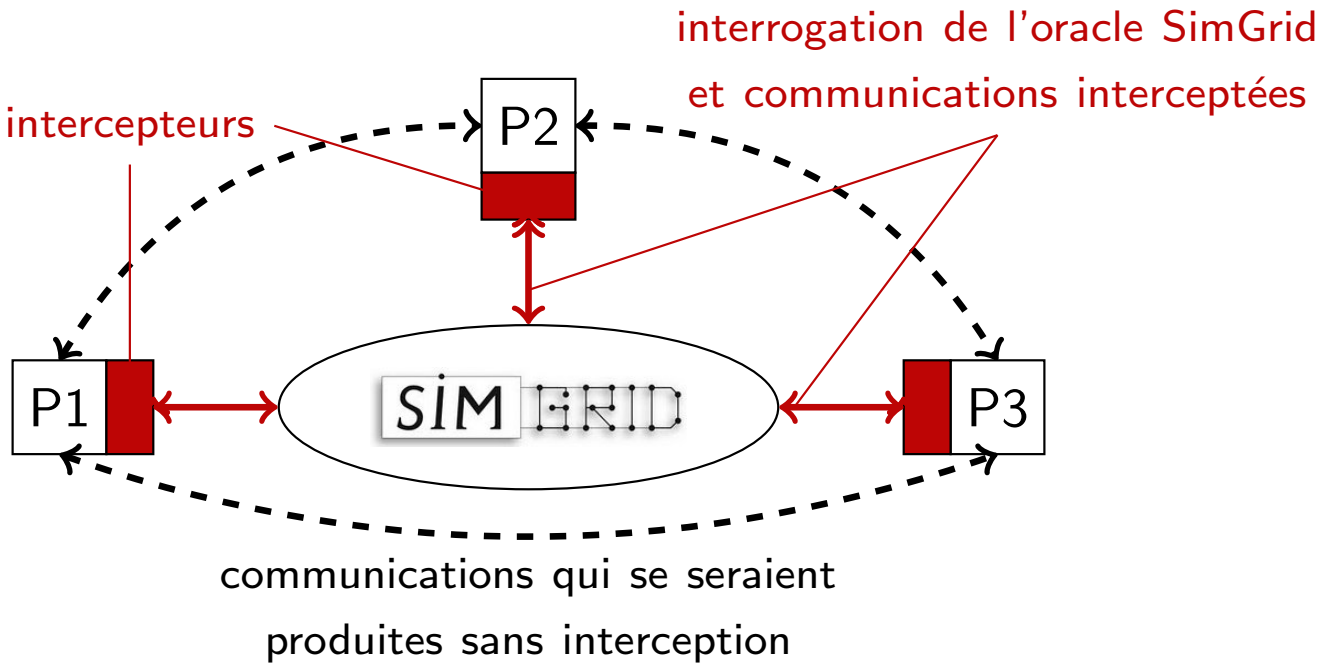
- Ne dépend pas d'une infrastructure complexe
- Déployable sur un ordinateur portable ou un cluster
- Permettant le **repliement de processus** sous un **large éventail de conditions** et **l'étude du comportement** de l'application **pendant son exécution**

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 très 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[1]

1. http ://simgrid.gforge.inria.fr/

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 : 2 approches
   - Sans exécuter réellement l'opération : calcul de la durée de chaque action sur la plate-forme virtuelle par le simulateur
   - Exécution de l'opération en mesurant sa durée et application 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 : 2 approches
   - Sans exécuter réellement l'opération : calcul de la durée de chaque action sur la plate-forme virtuelle par le simulateur
   - Exécution de l'opération en mesurant sa durée et application 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

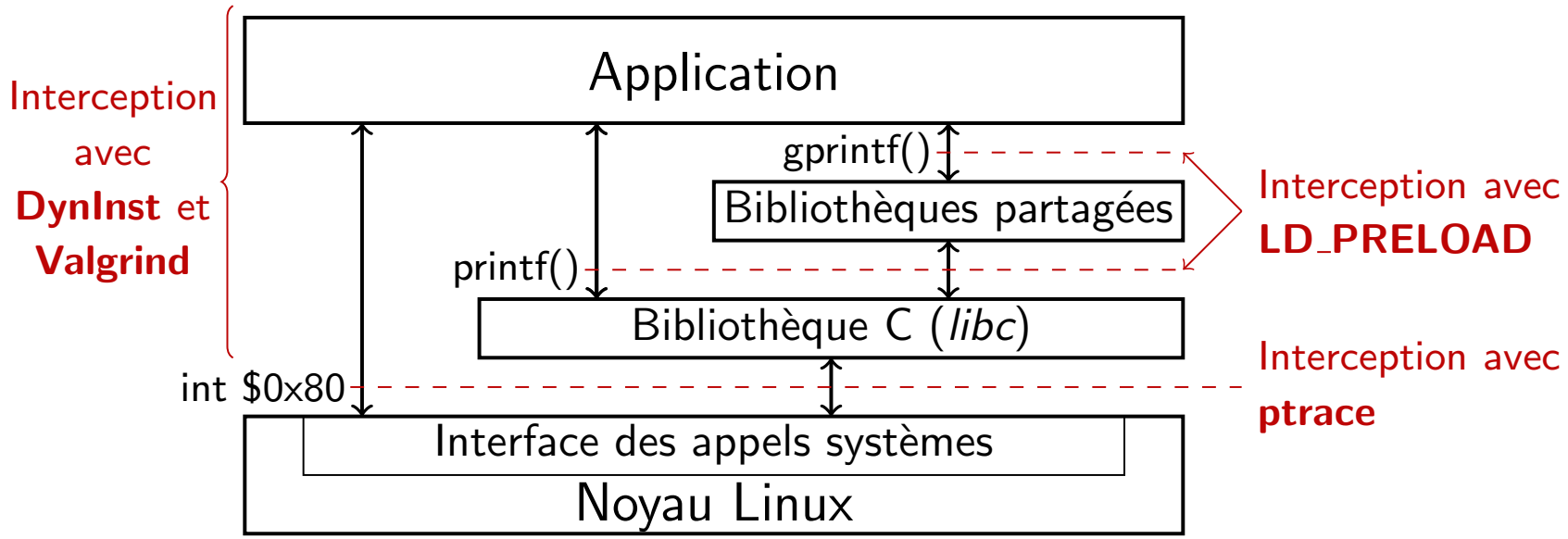
1. Sélection des actions à intercepter
   - Actions liées à la création et à l'identité des processus
   - Actions liées aux entrées/sorties et aux communications
   - Actions liées au temps : travail en temps réel ou virtualisation du temps
2. Reproduire l'impact de la plate-forme virtuelle sur l'exécution : 2 approches
   - Sans exécuter réellement l'opération : calcul de la durée de chaque action sur la plate-forme virtuelle par le simulateur
   - Exécution de l'opération en mesurant sa durée et application 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 :



- Développement d'un prototype pour chaque méthode

Valgrind et DynInst

Valgrind : outil de débogage et de 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
  ⇒ 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 débogage et de 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

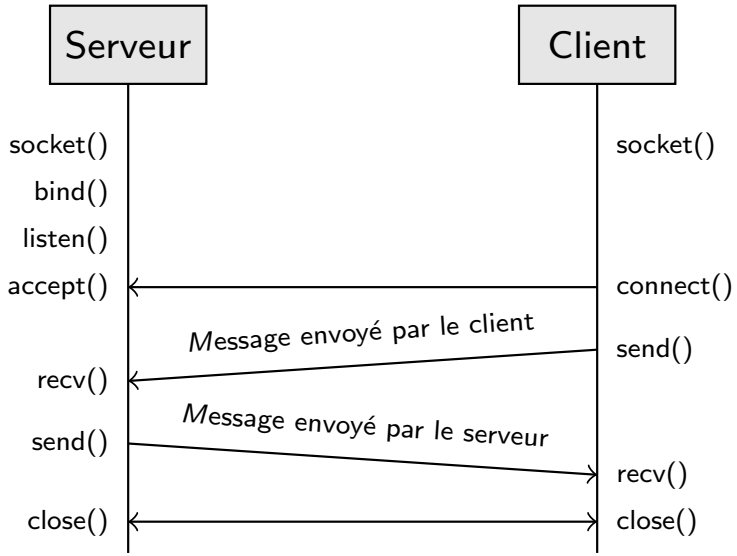
1. Principe similaire à l'utilitaire `strace`
   - Extraction des appels systèmes et des paramètres pour chaque processus
   - Identification des processus communiquants

```
..
[24402] getsockopt(4, SOL_SOCKET, SO_REUSEADDR, 1 ) = 0
[24402] bind( 4, {sa_family=AF_INET, sin_port=htons(2226), sin_addr=inet_addr("0.0.0.0")}, 16 ) = 0
[24419] connect( 4, {sa_family=AF_INET, sin_port=htons(2226), sin_addr=inet_addr("127.0.0.1")}, 16 )
      = 0
[24402] accept( 4, {sa_family=AF_INET, sin_port=htons(56842), sin_addr=inet_addr("127.0.0.1")}, 16 )
      = 5
..
[24419] exit_group(0) called
```

2. Détermination des périodes de calcul

3. Validation sur une application pair-à-pair réelle : BitTorrent

Exemple de trace client/serveur



pidX	syscall	pidY	wall_time	cpu_time	diff_wall	diff_cpu	local_addr:port	remote_addr:port	return	param
6976	(v)fork		19234	12000	0	12000			6977	
6976	(v)fork		25537	16000	6303	4000			6978	
6976	(v)fork		3038838	16000	3013301	0			6989	
...										
6977	recv		3031988	0	0	0	127.0.0.1: 2226	127.0.0.1:34024		
6989	send	6977	12697	0	0	0	127.0.0.1:34024	127.0.0.1: 2226		
6989	send	6977	12895	0	198	0	127.0.0.1:34024	127.0.0.1: 2226	512	(4,"...",512)
6977	recv	6989	3032640	0	652	0	127.0.0.1: 2226	127.0.0.1:34024	512	(5,"...",512)
6989	recv	6977	13133	0	238	0	127.0.0.1:34024	127.0.0.1: 2226		
6977	send	6989	3032963	0	323	0	127.0.0.1: 2226	127.0.0.1:34024		
6977	send	6989	3033136	0	173	0	127.0.0.1: 2226	127.0.0.1:34024	512	(5,"...",512)
6989	recv	6977	13643	0	510	0	127.0.0.1:34024	127.0.0.1: 2226	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 de l'application
  2. Identification des périodes de calculs
  3. 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 de la mécanique d'interception et de feedback
  - Démarrage automatique des nœuds sur la plate-forme simulée
- **Émulation distribuée** : distribution des processus sur différentes machines
  - Utilisation d'un cluster
  - Outil intégré pour l'étude d'applications réelles sur plates-formes simulées