Vol de Tâches Efficace pour Systèmes Événementiels Multi-cœurs

Fabien Gaud, Sylvain Genevès, Fabien Mottet

INRIA Rhône-Alpes
Université de Grenoble
Équipe SARDES

CFSE'7

Outline

Objectifs

- Contexte : Modèle de programmation événementiel
- Applications ciblées : serveurs de données

- Les architectures multi-cœurs sont de plus en plus répandues
  - Dans nos tests nous utilisons un serveur bi-processeur double-cœur
- Il est donc nécessaire d'utiliser efficacement ces nouvelles architectures pour obtenir de bonnes performances

$\Rightarrow$ Nous voulons proposer un support d'exécution efficace pour la programmation événementielle sur les architectures multi-cœurs

Support d'exécution événementiel

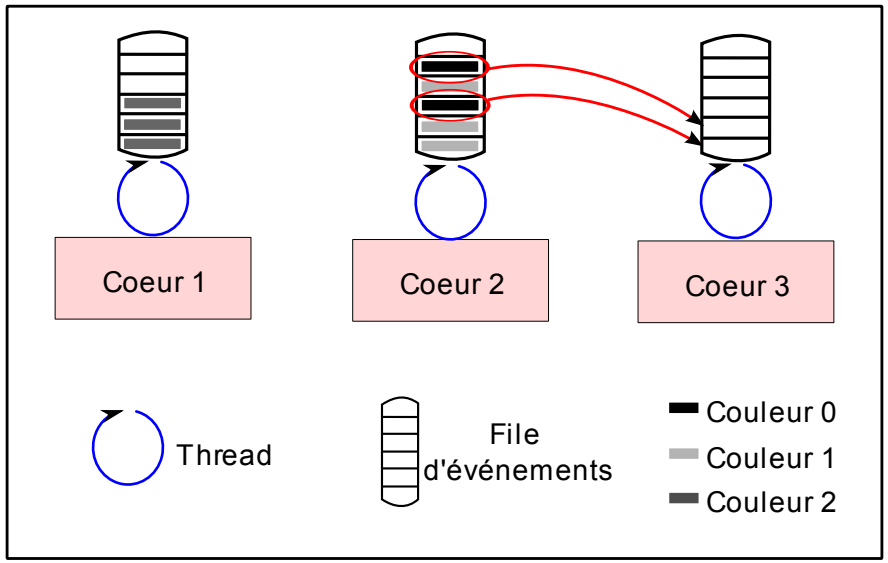
- Application composée d'événements et de traitants d'événements
- Un *thread* traite les événements de sa file
  - Récupère un événement dans la file d'événements
  - Appelle le traitant associé
- Le support multi-cœur consiste essentiellement à répartir les événements sur les cœurs

Support d'exécution événementiel multi-cœurs

- SEDA
  - Une architecture hybride (les traitants ont leurs propres ressources d'exécution)
  - Ne s'intéresse pas à la répartition des événements
- SMP Click
  - Modèle de programmation et support d'exécution pour construire des routeurs performants
  - La répartition de charge est contrainte par le modèle de programmation
- Libasync-SMP
  - Conserve le modèle de programmation événementiel classique
  - La répartition de charge est basée sur le mécanisme de vol de tâches

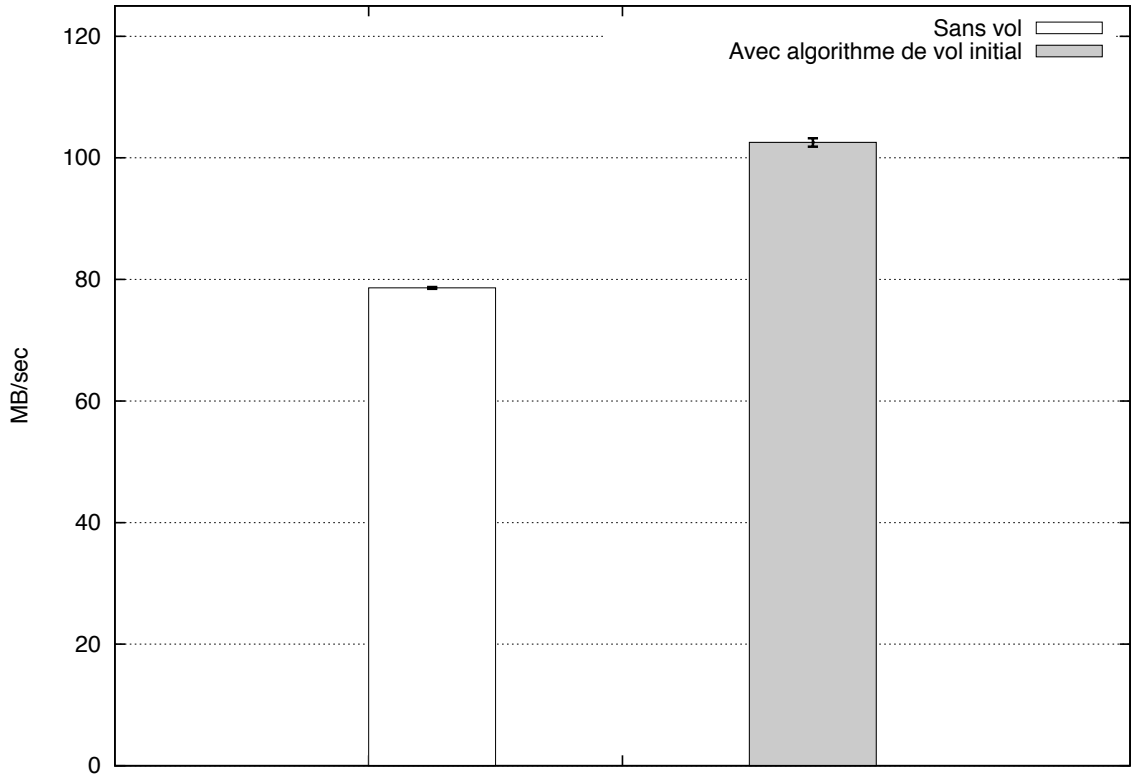
Libasync-SMP

- Une file d'événements par cœur
- L'exclusion mutuelle est assurée en coloriant les événements
  $\Rightarrow$ Le support d'exécution assure que deux événements de la même couleur ne seront pas exécutés simultanément
- Répartition des événements
  - Les couleurs sont réparties de façon tournante
  - La répartition de charge est réévaluée constamment grâce un algorithme de vol de tâches



Comportement espéré : le cas SFS (Secured File Server)

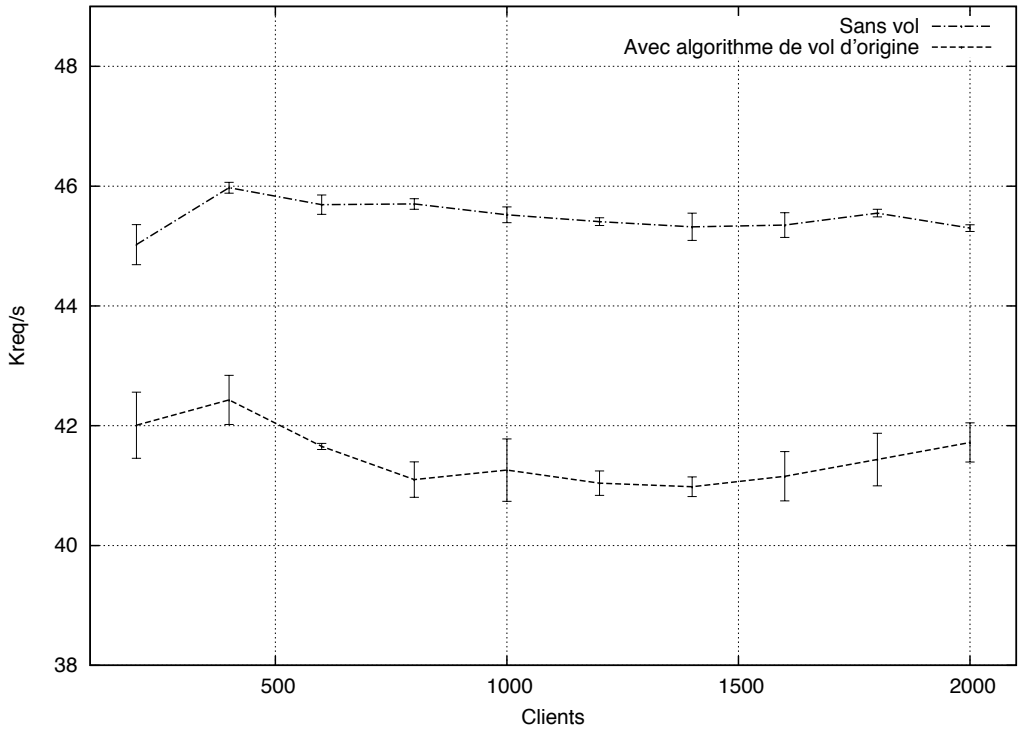
- Beaucoup d'opérations cryptographiques
- Un bon cas d'étude pour le vol de tâches
- Cas d'étude : 15 clients accèdant au même fichier de 200Mo



⇒ 30% d'amélioration de performance due au vol de tâches

Comportement non voulu : le cas du serveur Web

- Serveur web servant des pages statiques
- Les coûts du mécanisme de vol de tâches sont importants
- Cas d'étude : clients accédant à des fichiers de 1K



⇒ 10% de baisse de performances due au vol de tâches

Outline

Idée #1 : Réduire les coûts de verrouillage

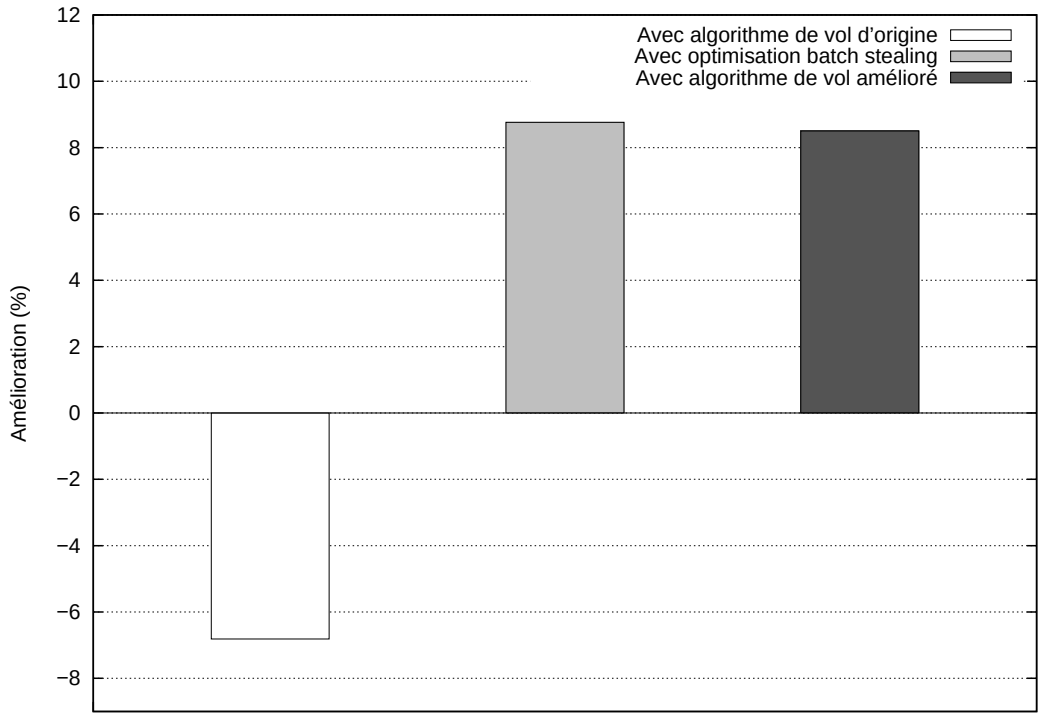
- Les verrous sont pris lors de la manipulation des files d'événements

- Le traitement d'un événement peut être très court
- La plupart des vols ne déplacent qu'un petit nombre d'événements $\Rightarrow$ vols fréquents

- Idée : réduire la fréquence de vols en augmentant la quantité d'événements volés
- Batch stealing $\Rightarrow$ voler plusieurs couleurs à la fois

Idée #1 : Évaluation

- Cas d'étude : application composée de 5 traitants successifs
- La taille des événements est négligeable, la durée du traitement est très faible



⇒ 18% d'amélioration par rapport au vol d'origine

Idée #1 : Impact sur les coûts de verrouillage

Sans vol de tâches	1,2%
Avec algorithme de vol d'origine	15,4%
Avec optimisation *batch stealing*	5,1%
Avec algorithme de vol amélioré	5,8%

FIG.: Pourcentage de temps passé dans les prises de verrous

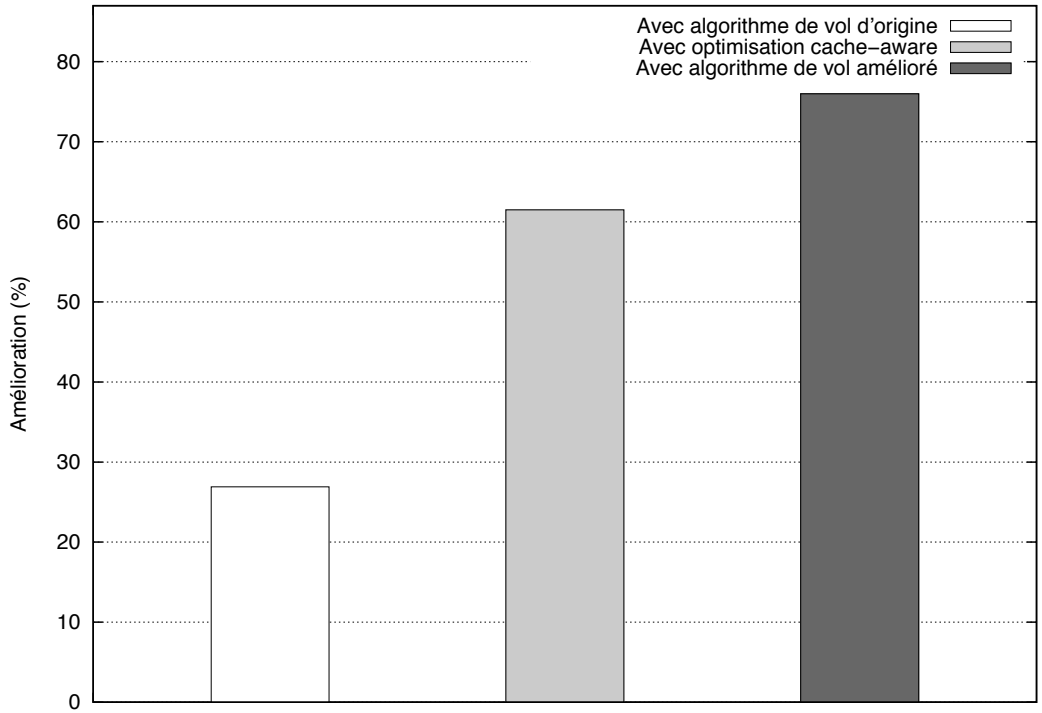
Idée #2 : Améliorer l'utilisation des caches

- Dans les architectures multi-cœurs, les cœurs partagent un (ou plusieurs) cache(s)
- Les temps d'accès à un cache sont significativement meilleurs que ceux à la mémoire centrale

- Idée : Prendre la hiérarchie de cache en considération lors d'une tentative de vol
- Cache-efficient stealing $\Rightarrow$ Donne la priorité au vol sur les cœurs *voisins*

Idée #2 : Évaluation

- Cas d'étude : application composée de 5 traitants successifs
- Les messages échangés ont une taille importante et sont manipulés par les traitants



⇒ 27% d'amélioration par rapport au vol de tâches d'origine

Idée #2 : Impact sur les fautes de caches L2

Avec algorithme de vol d'origine	71%
Avec optimisation *cache-aware*	44%
Avec algorithme de vol amélioré	36%

FIG.: Pourcentage de défauts de cache L2

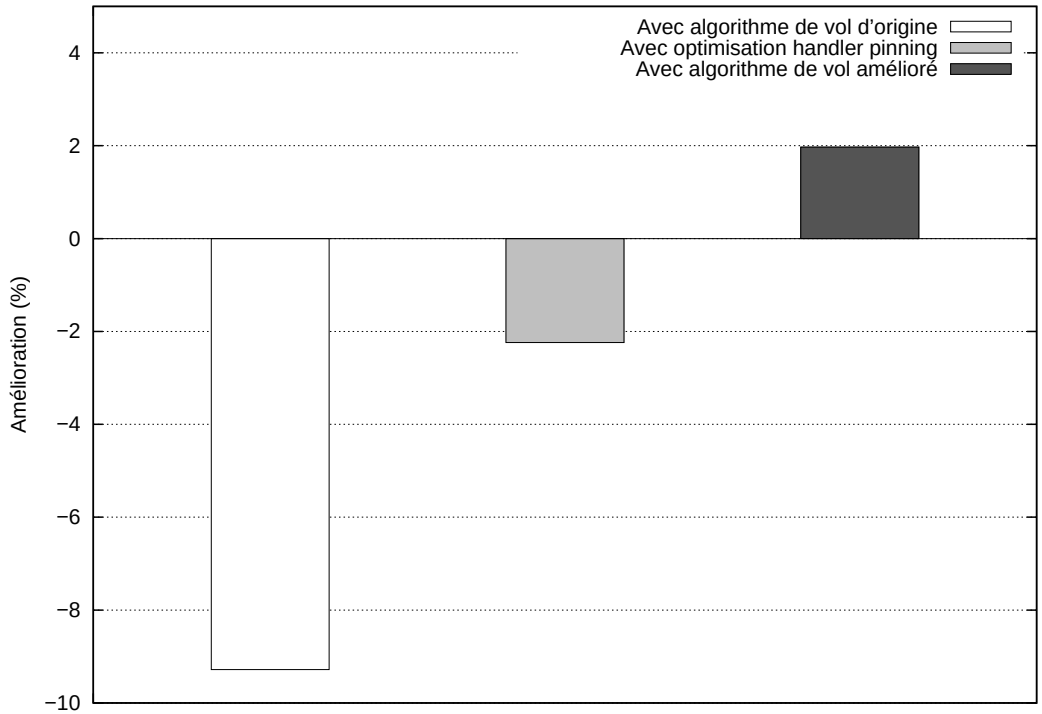
Idée #3 : Assigner un traitant à un cœur

- Le vol de tâches déplace tous les événements sans distinction
- Parfois, il peut être intéressant d'assigner un traitant à un cœur donné
  - Lorsque ce traitant manipule toujours une même donnée de taille importante
  - Lorsque les événements associés sont cruciaux pour assurer une charge suffisante

- Idée : Laisser au programmeur la possibilité d'assigner un type d'événements à un cœur

Idée #3 : Évaluation

- Cas d'étude : application composée de 4 traitants successifs
- Coloration par connexion
- Les événements liés à l'acceptation de la connexion sont fixés
- La taille des requêtes et des réponses sont négligeables



⇒ 8% d'amélioration comparé au vol de tâches d'origine

Idée #3 : Taux d'appel du traitant d'acceptation de connexion

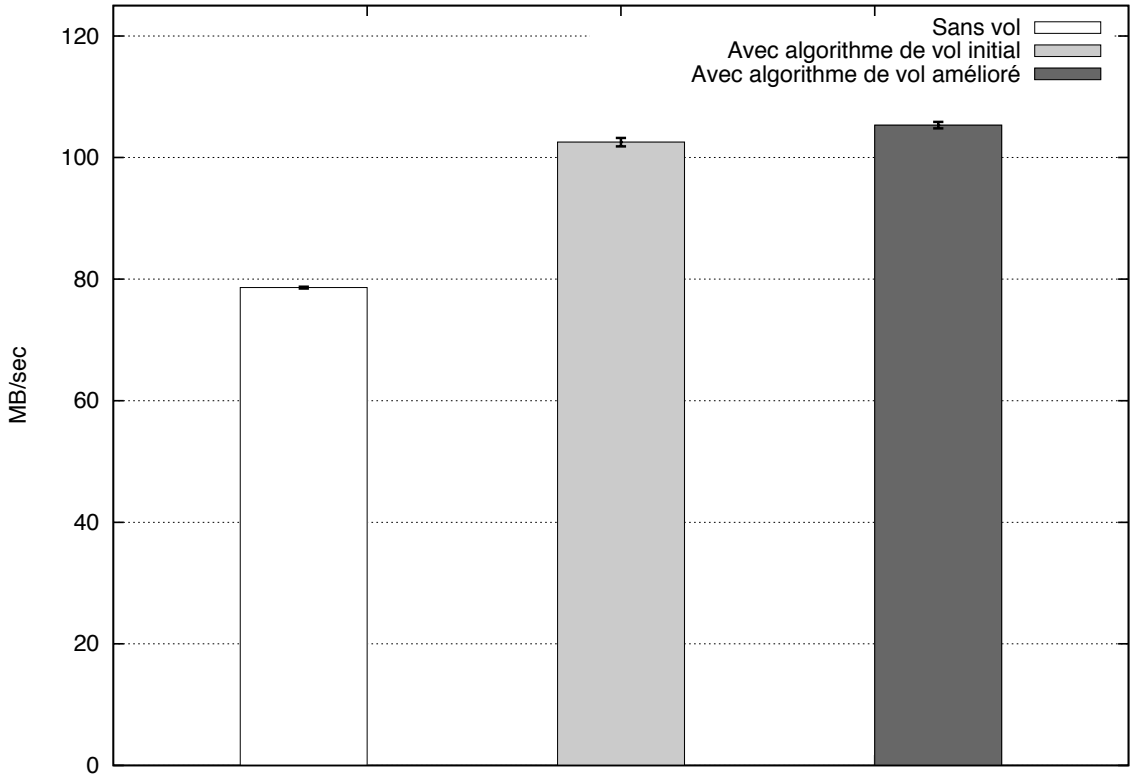
Avec algorithme de vol d'origine	-9%
Avec optimisation *handler pinning*	+2%
Avec algorithme de vol amélioré	+2%

FIG.: Augmentation du taux d'acceptation de connexion par rapport à la version sans vol de tâches

Outline

Évaluation sur SFS

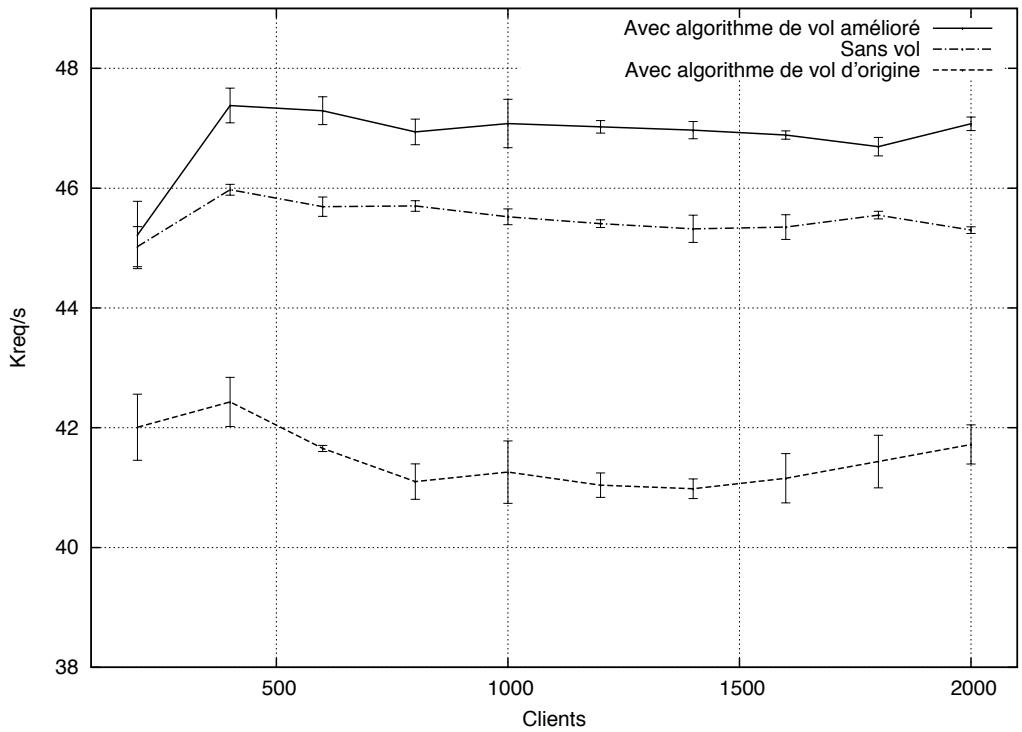
- 15 clients qui accèdent simultanément à un fichier de 200Mo
- 60% du temps passé dans les opérations de chiffrage $\Rightarrow$ seules ces opérations sont coloriées



$\Rightarrow$ comme attendu seulement une petite augmentation de performance par rapport au vol de tâches d'origine

Évaluation sur le serveur Web

- Retourne un contenu statique (fichier de 1Ko)
- Coloration par connexion / Traitant acceptant les connexions assignées au cœur 1
- 5 machines physiques simulant entre 200 et 2000 clients



⇒ jusqu'à 15% d'amélioration par rapport au vol de tâches d'origine

Outline

Conclusion

- Le vol de tâches permet de répartir la charge sur les cœurs
- Cependant, dégrade parfois les performances

- 3 améliorations proposées
  - Batch stealing
  - Cache-aware stealing
  - Handler pinning

- Améliorations obtenues à la fois sur des micro-tests et sur des applications réelles

Travaux futurs

- Consolider les résultats & améliorer les heuristiques
  - Différentes injections de charge
  - Passage à l'échelle sur 8 et 16 coeurs
  - Vol de tâches avec priorités

- Améliorer le support d'exécution
  - Gestion de la mémoire
  - Gestion des communication inter-cœurs

Questions ?