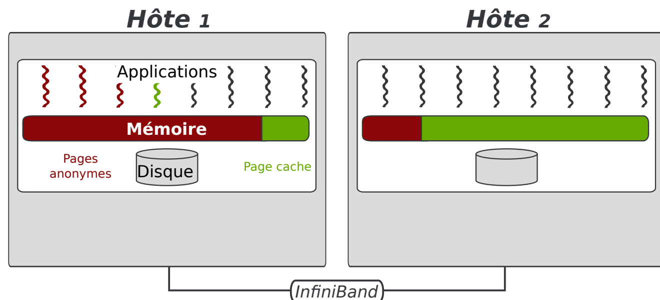
Puma : Un cache distant pour mutualiser la mémoire inutilisée des machines virtuelles

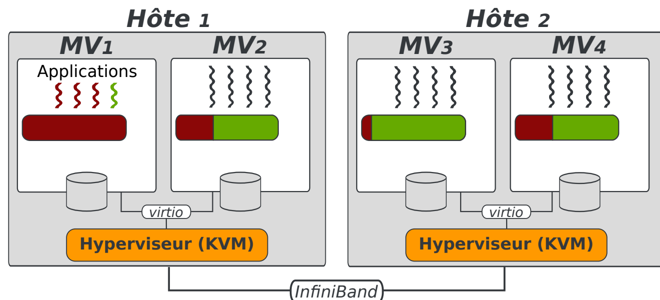
Maxime Lorrillere, Julien Sopena, Sébastien Monnet and Pierre Sens

24 avril 2014

Problème : fragmentation de la mémoire

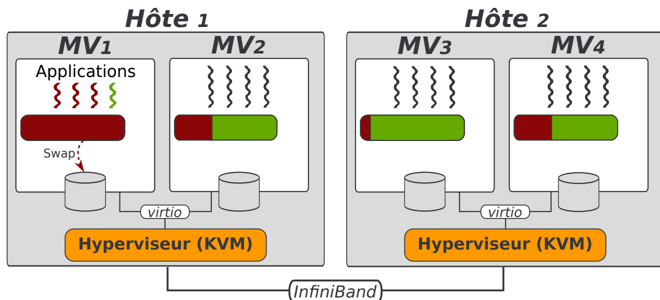


Problème : fragmentation de la mémoire



- La virtualisation apporte flexibilité et isolation

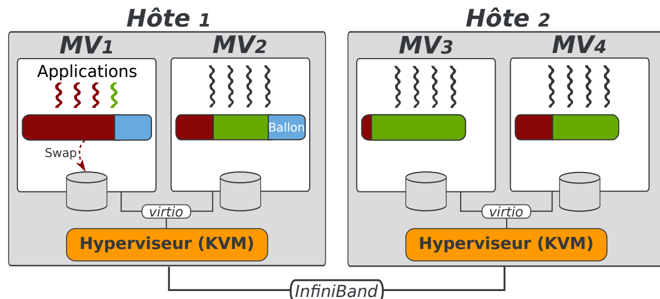
Problème : fragmentation de la mémoire



- La virtualisation apporte flexibilité et isolation
  Problème : fragmentation de la mémoire disponible
  - ⇒ Partager des ressources comme le temps CPU est facile
  - ⇒ Ajuster la quantité de mémoire d'une MV prend du temps et peut générer du *swap*

Solution : réallouer dynamiquement la mémoire

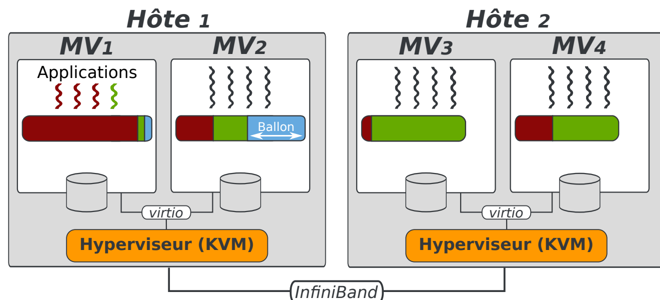
Memory Ballooning [OSDI'02]



- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Solution : réallouer dynamiquement la mémoire

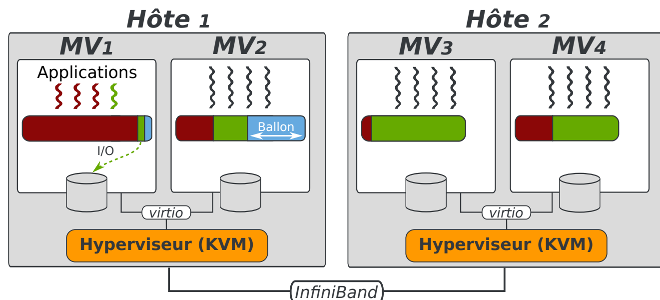
Memory Ballooning [OSDI'02]



- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Solution : réallouer dynamiquement la mémoire

Memory Ballooning [OSDI'02]



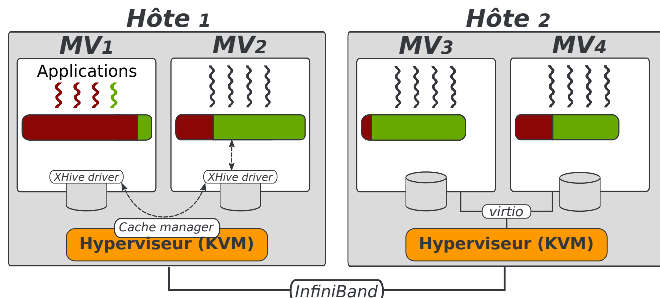
- Le ballon est *gonflé* pour rendre de la mémoire à l'hôte
- Le ballon est *dégonflé* pour reprendre de la mémoire
- L'ensemble est coordonné par l'hôte

Problème : l'hôte ne sait pas à quoi sert la mémoire

⇒ Entraîne une fragmentation du page cache

Solution : cache coopératif virtualisé

XHive [IEEE TOC'11]

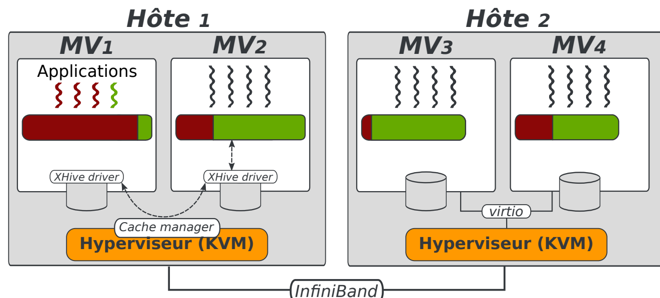


Cache coopératif entre les machines virtuelles :

- Repose sur un périphérique bloc virtuel spécifique
- Les I/O peuvent être servies depuis la mémoire des autres MV
- Un gestionnaire de cache est implanté dans l'hyperviseur (Xen)

Solution : cache coopératif virtualisé

XHive [IEEE TOC'11]



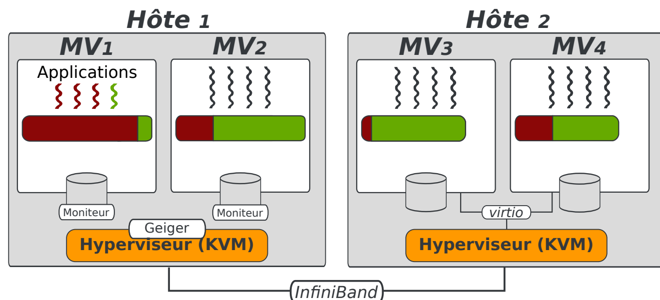
Cache coopératif entre les machines virtuelles :

- Repose sur un périphérique bloc virtuel spécifique
- Les I/O peuvent être servies depuis la mémoire des autres MV
- Un gestionnaire de cache est implanté dans l'hyperviseur (Xen)

Problèmes : incompatibilité avec les DFS, co-localisation des MV, modifications de l'hyperviseur

Solution : monitorer l'activité du page cache

Geiger [ASPLOS'06]

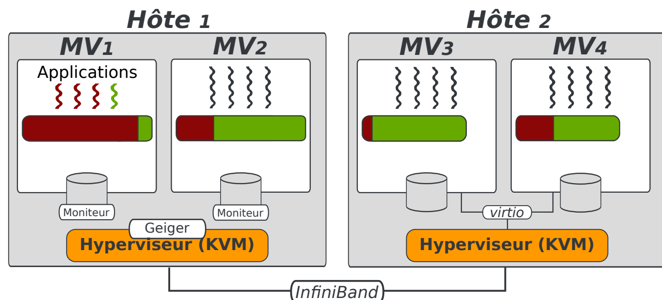


Objectif : détecter les manipulations du *page cache* sans modifier la MV

- Monitore les I/O et accès mémoire de la MV
- *Geiger* peut être utilisé pour créer un cache de second niveau

Solution : monitorer l'activité du page cache

Geiger [ASPLOS'06]



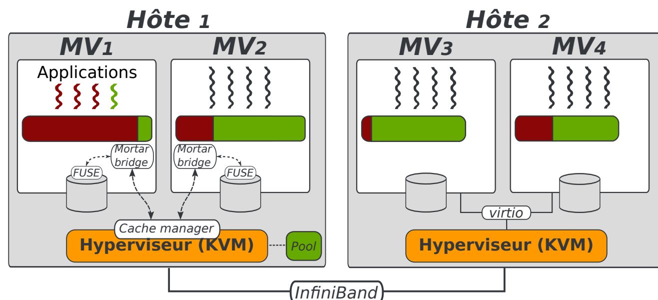
Objectif : détecter les manipulations du *page cache* sans modifier la MV

- Monitore les I/O et accès mémoire de la MV
- *Geiger* peut être utilisé pour créer un cache de second niveau

Problèmes : incompatibilité avec les DFS, modifications de l'hyperviseur

Solution : confier la gestion du cache à l'hyperviseur

Mortar [VEE'14]

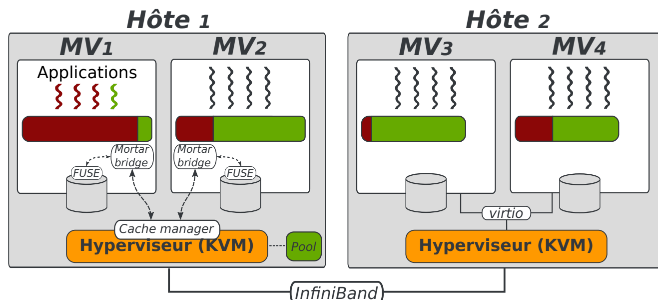


Framework pour réutiliser la mémoire inactive :

- Fourni des indictions à l'hôte sur la mémoire récupérable.
- L'hôte gère un pool de mémoire.
- API *Key/value store* compatible memcached.
- *MortarLoad* utilise cette API pour fournir un cache disque.

Solution : confier la gestion du cache à l'hyperviseur

Mortar [VEE'14]

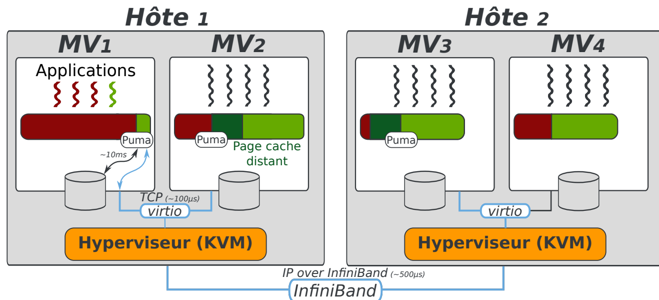


Framework pour réutiliser la mémoire inactive :

- Fourni des indictions à l'hôte sur la mémoire récupérable.
- L'hôte gère un pool de mémoire.
- API *Key/value store* compatible memcached.
- *MortarLoad* utilise cette API pour fournir un cache disque.

Problème : co-localisation des MV, modifications hyperviseur/MV

Notre solution : s'intégrer directement dans le *page cache*

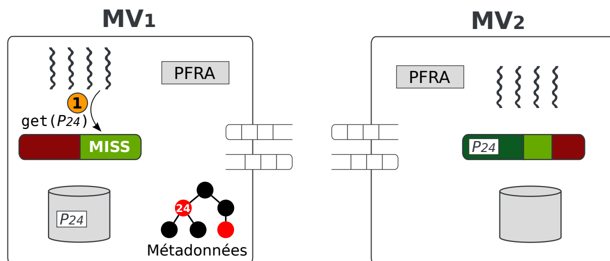


L'approche de Puma :

- Ne pas toucher à l'hyperviseur
  - ⇒ Intégration au noyau de la MV
- Ne dépend pas des périphériques blocs
  - ⇒ Supporte les *DFS*
- Approche similaire aux cache coopératifs

Architecture de Puma

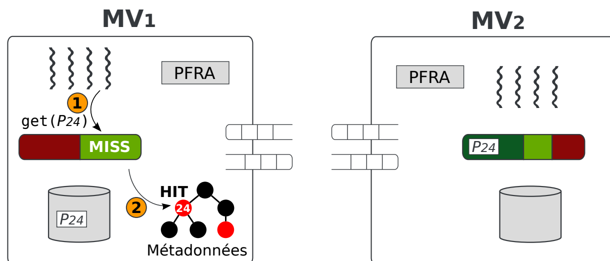
Défaut de page – opération *get*



- Intégration dans le *page cache* pour détecter les *miss*
- Un *miss* entraîne une opération *get* (synchrone)
- Méta-données locales pour éviter un *miss distant*

Architecture de Puma

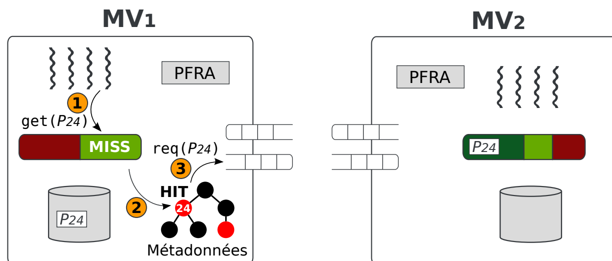
Défaut de page – opération *get*



- Intégration dans le *page cache* pour détecter les *miss*
- Un *miss* entraîne une opération *get* (synchrone)
- Méta-données locales pour éviter un *miss distant*

Architecture de Puma

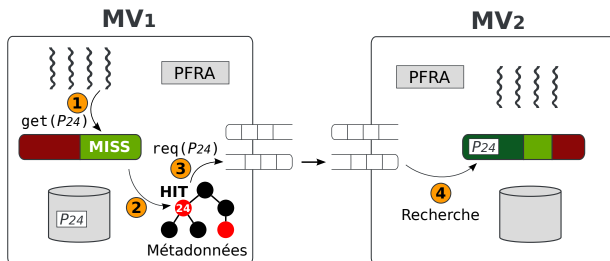
Défaut de page – opération *get*



- Intégration dans le *page cache* pour détecter les *miss*
- Un *miss* entraîne une opération *get* (synchrone)
- Méta-données locales pour éviter un *miss distant*

Architecture de Puma

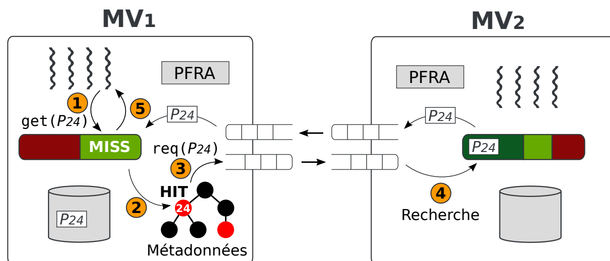
Défaut de page – opération *get*



- Intégration dans le *page cache* pour détecter les *miss*
- Un *miss* entraîne une opération *get* (synchrone)
- Méta-données locales pour éviter un *miss distant*

Architecture de Puma

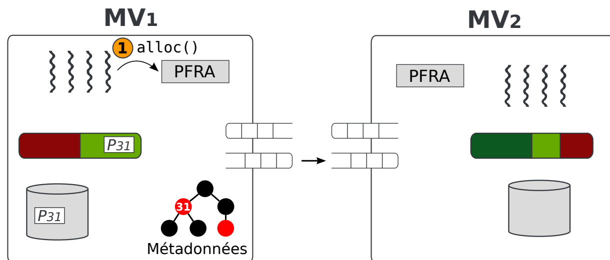
Défaut de page – opération *get*



- Intégration dans le *page cache* pour détecter les *miss*
- Un *miss* entraîne une opération *get* (synchrone)
- Méta-données locales pour éviter un *miss distant*

Architecture de Puma

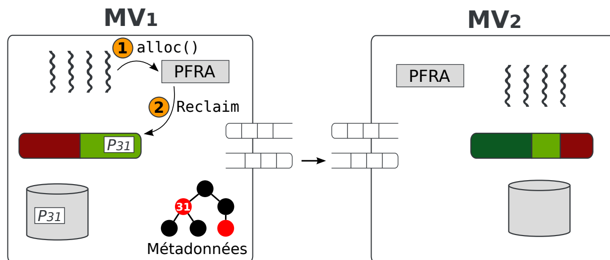
Éviction d'une page – opération *put*



- Généralement déclenchée par une allocation de mémoire
- Intégration dans le *PFRA* pour détecter les évictions
- Traitement asynchrone pour éviter les ralentissements

Architecture de Puma

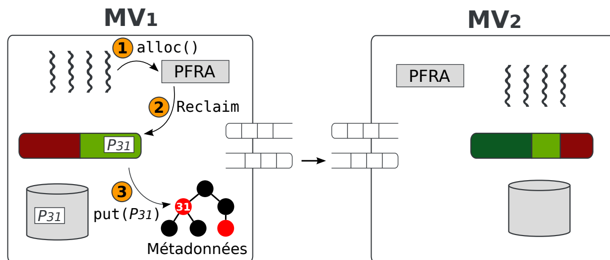
Éviction d'une page – opération *put*



- Généralement déclenchée par une allocation de mémoire
- Intégration dans le *PFRA* pour détecter les évictions
- Traitement asynchrone pour éviter les ralentissements

Architecture de Puma

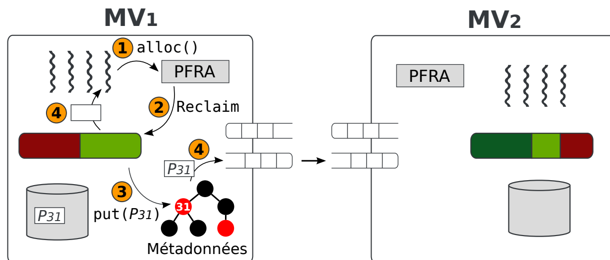
Éviction d'une page – opération *put*



- Généralement déclenchée par une allocation de mémoire
- Intégration dans le *PFRA* pour détecter les évictions
- Traitement asynchrone pour éviter les ralentissements

Architecture de Puma

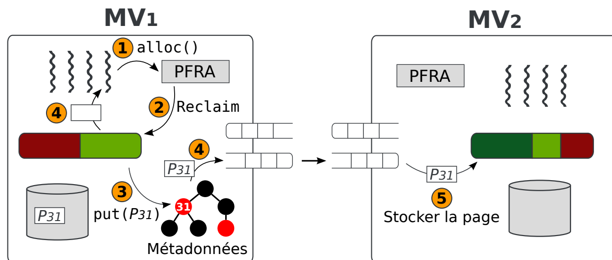
Éviction d'une page – opération *put*



- Généralement déclenchée par une allocation de mémoire
- Intégration dans le *PFRA* pour détecter les évictions
- Traitement asynchrone pour éviter les ralentissements

Architecture de Puma

Éviction d'une page – opération *put*



- Généralement déclenchée par une allocation de mémoire
- Intégration dans le *PFRA* pour détecter les évictions
- Traitement asynchrone pour éviter les ralentissements
- Les pages sont stockées en mémoire dans une *LRU*
- Stratégies de cache
  - ⇒ Exclusive et inclusive

Détails et optimisations

Empreinte mémoire

$\Rightarrow$ Méta-données : 60 o/page, soit 16 Mo/Go de cache

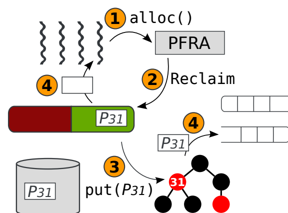
Détails et optimisations

Empreinte mémoire

⇒ Méta-données : 60 o/page, soit 16 Mo/Go de cache

Gestion de la mémoire : éviter les interblocages

- Évite des recopies dans la couche réseau
- Allocations de mémoire limitées et atomiques
- *Pools* de mémoire pré-allouée



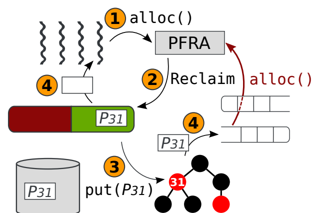
Détails et optimisations

Empreinte mémoire

⇒ Méta-données : 60 o/page, soit 16 Mo/Go de cache

Gestion de la mémoire : éviter les interblocages

- Évite des recopies dans la couche réseau
- Allocations de mémoire limitées et atomiques
- *Pools* de mémoire pré-allouée



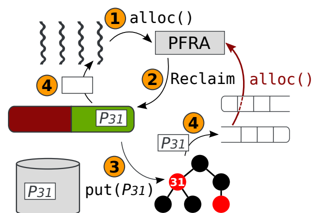
Détails et optimisations

Empreinte mémoire

⇒ Méta-données : 60 o/page, soit 16 Mo/Go de cache

Gestion de la mémoire : éviter les interblocages

- Évite des recopies dans la couche réseau
- Allocations de mémoire limitées et atomiques
- *Pools* de mémoire pré-allouée



Gestion de la cohérence

- Les pages « sales » sont d'abord écrites sur disque
- Une page modifiée localement est invalidée (stratégie inclusive)

État actuel

- Prototype fonctionnel
- Indépendent des périphérique blocs et des systèmes de fichiers
- Essentiellement sous forme de module noyau (6000 lignes)
- Quelques modifications au cœur du noyau ($\pm$ 300 lignes)
- Évolution facile vers les noyaux récents (3.11.6)

Plateforme expérimentale

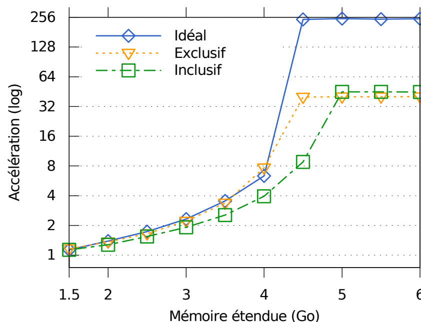
Plateforme virtualisée (*KVM*)

- Serveur de cache : 2 vCPU, fourni de 512 Mo à 5 Go de cache
- Client : 2vCPU, 1 Go
- Point de référence : une MV avec 1 Go de mémoire
- Quantité totale de cache : client + serveur (1.5 Go à 6 Go)

Types d'expérimentations

1. Avec des MV co-localisées et un réseau paravirtualisé (*virtio*)
2. Avec des MV distantes et un réseau *InfiniBand* (*IPoIB*)
3. Comparaison avec les performances d'un cache SSD

Microbenchmarks en lecture seule

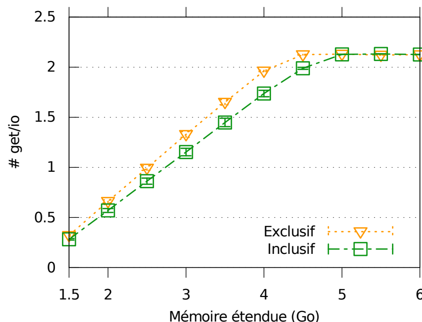


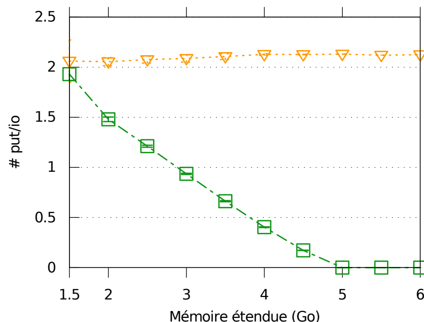
Lecture de blocs de 4 Ko dans un fichier de 4 Go :

- Gain de performances de +15% avec 1.5 Go de cache
- x40 lorsque l'ensemble des données tiennent dans le cache
- La stratégie exclusive est plus efficace jusqu'à ce que l'ensemble des données tiennent dans le cache

Microbenchmarks en lecture seule

Stratégies de cache





Startégie exclusive Fourni plus de cache que le cache inclusif

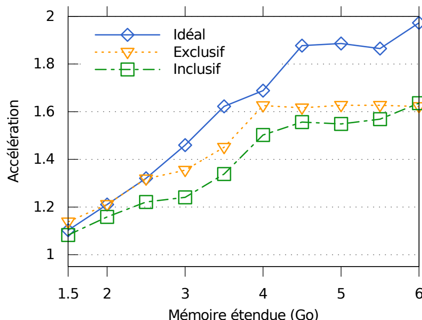
$\Rightarrow$ Taux de *hit* plus élevé

Stratégie inclusive Conserve les pages dans le cache

$\Rightarrow$ Permet d'éviter de les renvoyer au serveur

Benchmarks applicatifs

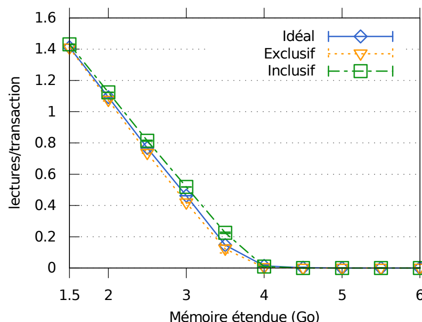
Postmark – Performances

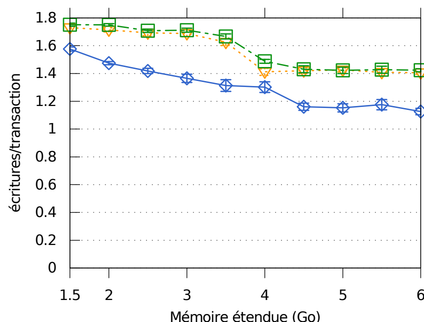


- Configuration défavorable : 80% d'écritures, 20% de lectures
- Au minimum, une amélioration des performances de +10%
- Gain de +70% lorsque l'essentiel des données tiennent dans le cache

Benchmarks applicatifs

Postmark – E/S

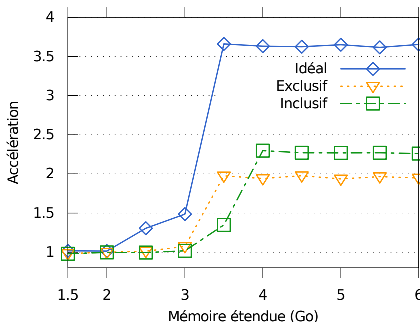




- Toutes les lectures finissent dans le cache (local ou distant)
- L'ajout de cache pour les pages « propres » laisse plus de place pour les écritures différées

Benchmarks applicatifs

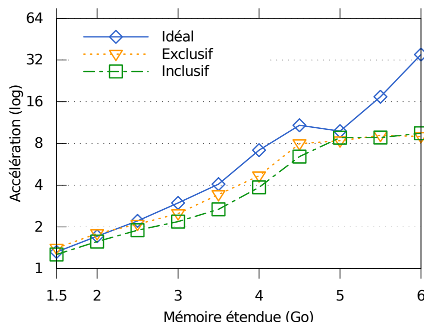
Blast



Base de données de séquences de 3 Go :

- Accès séquentiels : baisse de performances (2%)
  - $\Rightarrow$ Peut être résolu via une politique de remplacement mieux adaptée côté serveur
- x2 lorsque la base de données tient dans le cache

Benchmarks applicatifs
TPC-H

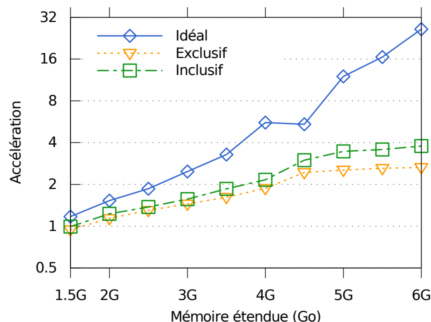


- Répartition favorable des écritures (90%/10%)
- Au minimum, 25% à 40% d'amélioration de performances
- x9 lorsque la base de données tient en mémoire

Benchmarks applicatifs

TPC-H sur *InfiniBand*

- Cluster *Graphene* de Grid'5000
- Les VM sont sur des nœuds dédiés
- Les nœuds sont interconnectés via *InfiniBand* (20 GB/s)



- Légère dégradation de performances (5%) avec peu de cache
- 20% de gain à partir de 2 Go de mémoire
- Le cache inclusif est plus efficace

Benchmarks applicatifs

Cache SSD avec *dm-cache*

Cache sur SSD :

1. VM co-localisées
2. SSD Samsung 840 Pro avec le module noyau *dm-cache*

	RR (IO/s)	Pm (T/s)	BLAST (s)	TPC-H (s)
Référence (1 Go)	166	64.7	42.3	7299.0
Idéal (6 Go)	40400	**219.2**	11.5	233.5
Puma (exclusive)	5885	104.6	21.2	910.0
Puma (inclusive)	**6310**	100.5	**17**	**861.8**
dm-cache	3250	**695.2**	21.3	1148.1

Conclusion

- Puma apporte une solution au problème de fragmentation du cache
- Prototype fonctionnel et non-intrusif
- Évaluation intensive :
  - Une amélioration allant jusqu'à x40 dans les microbenchmarks
  - Jusqu'à x9 dans des applications de type base de données

Perspectives

- Détection de l'(in)activité côté serveur
- Gestion d'écritures synchrones dans le cache