Puma: pooling unused memory in virtual machines for I/O intensive applications

Maxime Lorrillere, Julien Sopena, Sébastien Monnet, Pierre Sens

To cite this version:

HAL Id: hal-01154566
https://hal.archives-ouvertes.fr/hal-01154566
Submitted on 22 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Puma: Pooling Unused Memory in Virtual Machines for I/O intensive applications

Maxime Lorrillere, Julien Sopena, Sébastien Monnet and Pierre Sens
Sorbonne Universités, Université Pierre et Marie Curie, CNRS

Context: virtualization fragments available memory

Virtualization allows more flexibility and isolation
Problem: it fragments available memory
⇒ Memory cannot be reassigned as efficiently as CPU time
⇒ Unused memory (i.e. idle caches) is wasted

Existing solution: Memory Ballooning
⇒ Allows to dynamically resize VM’s memory
⇒ Cannot efficiently reclaim unused memory
⇒ Does not benefit of unused memory on other hosts
⇒ Slow to recover memory

Our approach: PUMA

⇒ Rely on a fast network between VMs and hosts
⇒ PUMA can reuse unused memory of VMs hosted on different hosts
⇒ Handles clean cache pages
⇒ Writes are generally non-blocking
⇒ Simple consistency scheme
⇒ Fast to recover memory!
⇒ Exclusive and non-inclusive caching strategies

PUMA design

⇒ 2 basics operations
  ⇒ get(): gets a page from the [remote] page cache
  ⇒ put(): sends a victim page to the remote page cache
⇒ Local metadata with small memory footprint
  ⇒ amortized 64 bits/page, 2 MB of metadata per GB of cache
⇒ Pages are directly stored into the existing page cache
  ⇒ Memory is reclaimed naturally
⇒ Sequential I/O are detected and filtered
  ⇒ Disk bandwidth > network bandwidth
⇒ Network latency monitoring
  ⇒ PUMA is throttled when the latency becomes too high

Evaluation

⇒ Experiment setup
  ⇒ Active VM: 1 GB memory
  ⇒ Inactive VM: 512 MB → 12 GB memory
  ⇒ Network latency injection with Netem [LCA’05]
  ⇒ Benchmarks: Filebench, BLAST, TPC-C, TPC-H, Postmark

⇒ Conclusion and Future Work
  ⇒ PUMA solves the page cache fragmentation problem
  ⇒ It is based on an efficient kernel-level remote caching mechanism
  ⇒ It handles clean cache pages to quickly recover the memory
  ⇒ It works with co-localised VMs and remote VMs
⇒ Ongoing work: detecting when a VM has unused memory
  ⇒ Our idea: toggle PUMA service based on Linux’s active/inactive LRUs activity