Letting Go

(A sensitive guide to dealing with application caches under the pressures of low memory)

John Stultz

August 30th 2012

http://www.linaro.org

Linaro

Overview

- Caching
- How mm/page management works
- Some different approaches managing caching
- Ideas for where these could be used
- Status

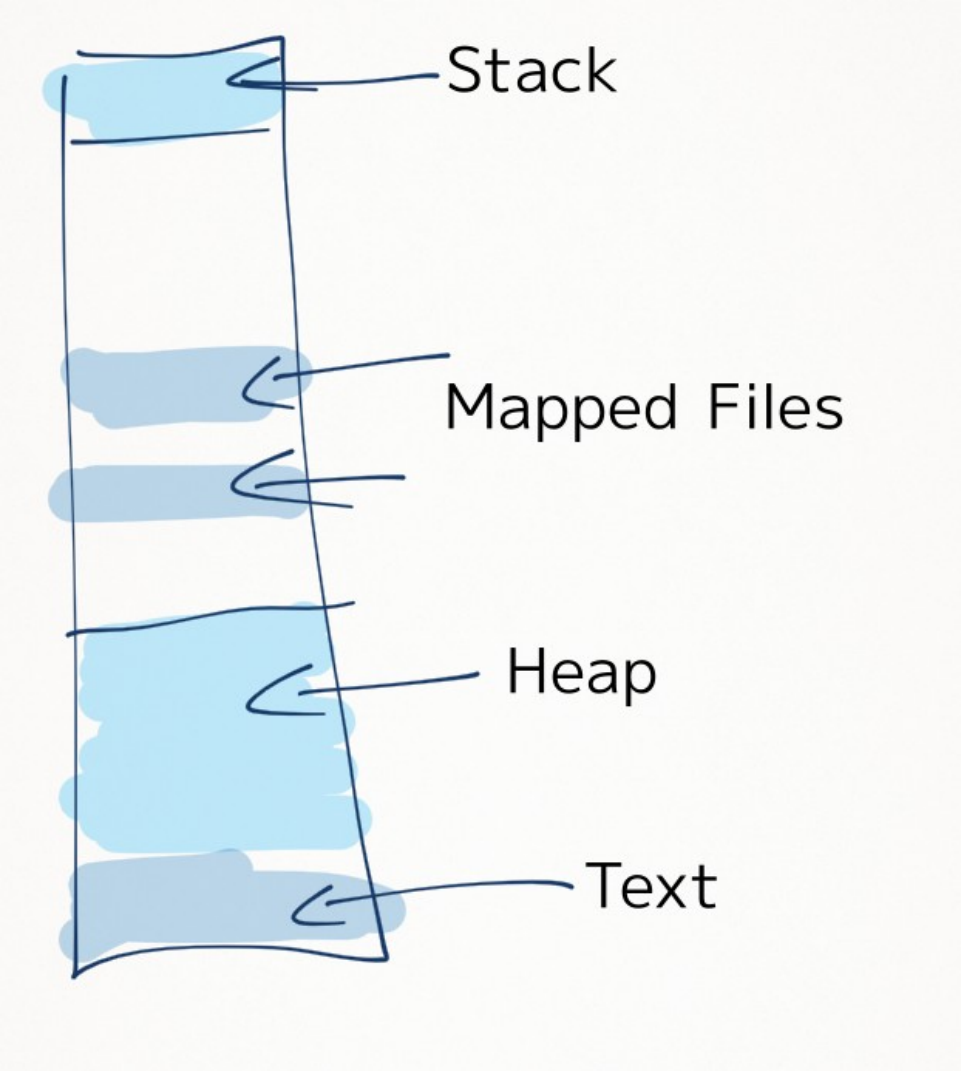
Caching

- Keeping data that is expensive to access around for future use
- Everyone is doing it!
  - Hardware
  - Kernel
  - Applications
- Not necessarily working together

When is it too much?

- Don't want to be a hoarder
- Memory used for caching cuts into memory used for other things
- How do we properly size application caches?

Bit of MM Background



Balancing two types of memory

- Anonymous pages
  - Heap
  - Stack
- File backed pages
  - Text data
  - Mapped files

Clean/Dirty

- Clean file pages are identical to the pages on disk.
  - Can be easily freed w/o any IO
- File pages that have been modified but not written back to disk are dirty
  - Takes extra time to write the data out before they are freed.

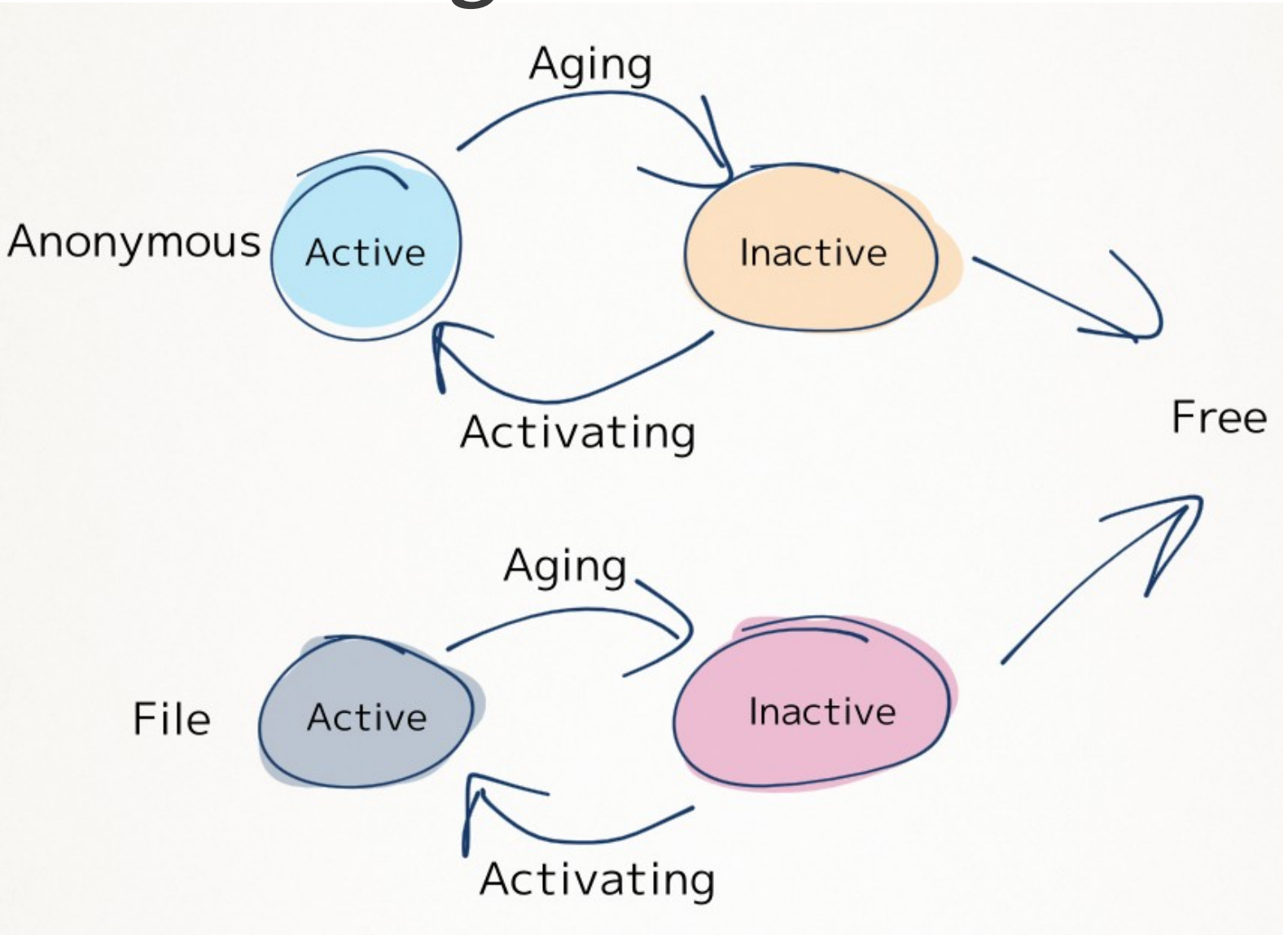
Free/Free-able

- Keep file pages around
- “Nothing is running, but memfree is low!”
  - Free vs Free-able
  - Free-able at what cost?

Page LRU lists

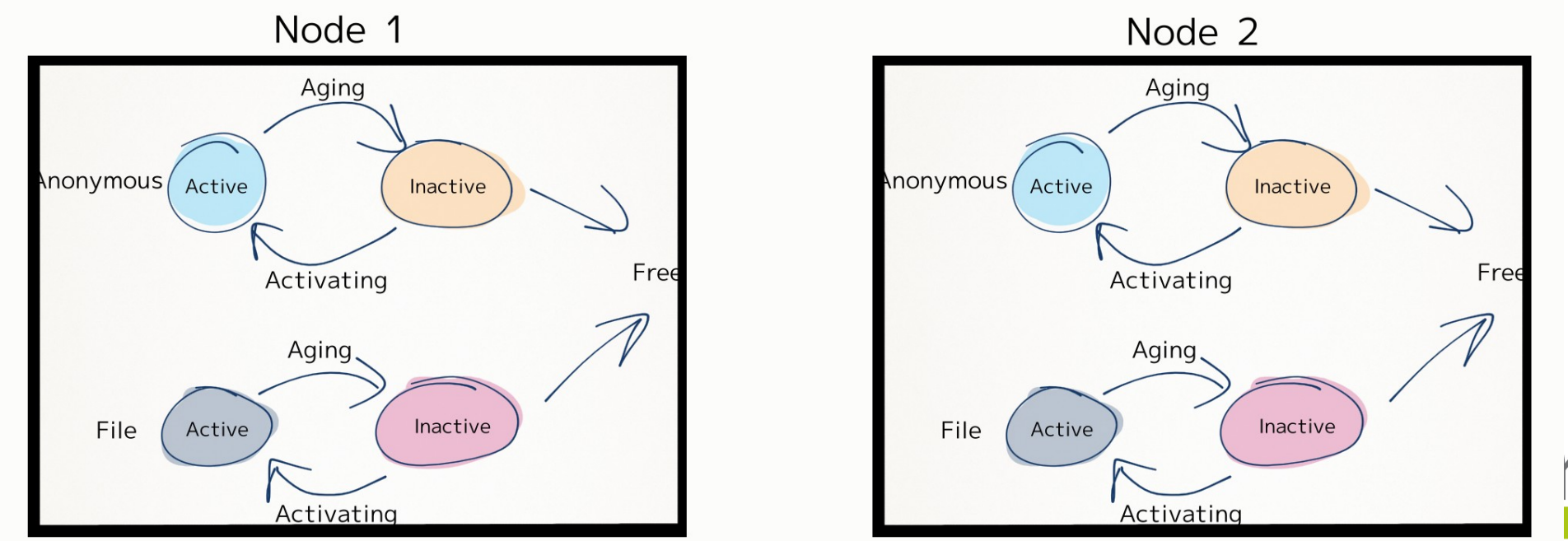
- For both Anonymous and File pages
  - Active and inactive LRU lists
  - If pages are touched, we move them to the active list
  - Periodically scan the active list, moving pages not recently touched to inactive list
  - Free pages from inactive list

Page LRU Lists



NUMA & Zones

- NUMA: Non-uniform-memory-access
  - CPUs and memory closer/further apart
- We use zones to manage group or nodes of memory
  - Have LRU lists and free lists for each zone



Slab Shrinkers

- Drivers and filesystems might have their own caches, which are separately managed from the kernels page-based mm.
- So for these, there are slab shrinkers, which allow filesystems and other drivers to free memory under pressure.
- Slab shrinkers are problematic though, because they are not numa aware.
- MM developers really don't like slab shrinkers.

Userland Caching

- Core problem:
  - Using a shared resource
  - difficult to size it properly.
- Solution: Let it go!

Letting Go

- Two philosophies
  - Manage it your self
  - Let the kernel do it for you (but give it good hints)
- Two proposed approaches
  - Low-memory notifications
  - Volatile ranges

Low Memory Notification

- Lets applications know when free pages in the system have dropped below a specified water level, and notifies when it has crossed back over that line when more pages are free
- Allows applications to make sure there are specified amount of memory available

Low Memory Killer

- Android™ has a low-memory-killer which is a in-kernel driver that watches memory levels and will kill specified applications before we hit critical OOM levels.
  - Avoids poor performance immediately before OOM
  - Uses slab shrinker
- Particularly useful with no swap

Low Memory Notification Interfaces

- Goal: Push low-memory-killer out to userland
- Memory-c-group (memcg) interface
  - Problematic memory/performance overhead
- Vmevents interface
  - By Pekka Enberg
  - Currently out-of-tree

Low Memory Notification Uses

- Applications could use these interfaces for cache management hints.
- Know when to shrink userland caches
- Particularly useful to free memory not in a classical cache
  - Applications, cached by keeping them running
  - A cache may be shared between many applications, thus some coordination might be necessary

Vmevent Issues

- Low free memory vs low freeable memory?
- Polling in userland vs polling in the kernel
- Some care needed here
  - If the system has already swapped out a cache, waiting for a low-memory notification to free cache pages could hurt performance.
  - Have to wait for pages to be swapped in, increasing memory usage, before they could be freed

Varnish Tangent

- Poul-Henning Kamp wrote an essay a while back
  - https://www.varnish-cache.org/trac/wiki/ArchitectNotes
  - 1975 programming vs 2006 programming
  - Instead of managing data in files and in memory, map the files into the applications address space and let the kernel manage whats on disk and whats in memory
  - Avoids “writing out” data that the kernel already swapped out

Volatile Ranges

- Came from Android's ashmem
- Abstract concept: mark inactive cache pages as “volatile”, which allows the kernel to reclaim them. Prior to use, mark pages “non-volatile”, and the kernel will tell you if it threw them away.
- Also can consider it like lazy & cancel-able hole punching

Volatile Ranges

- Goals: Avoid IO, but allow memory to be reclaimed.
- Most useful for data that can be regenerated faster then swapping pages in
- In the noswap case, gives kernel additional pages to reclaim
- Cost: Overhead of unmarking and marking pages when you're accessing them

Volatile Ranges Use Cases

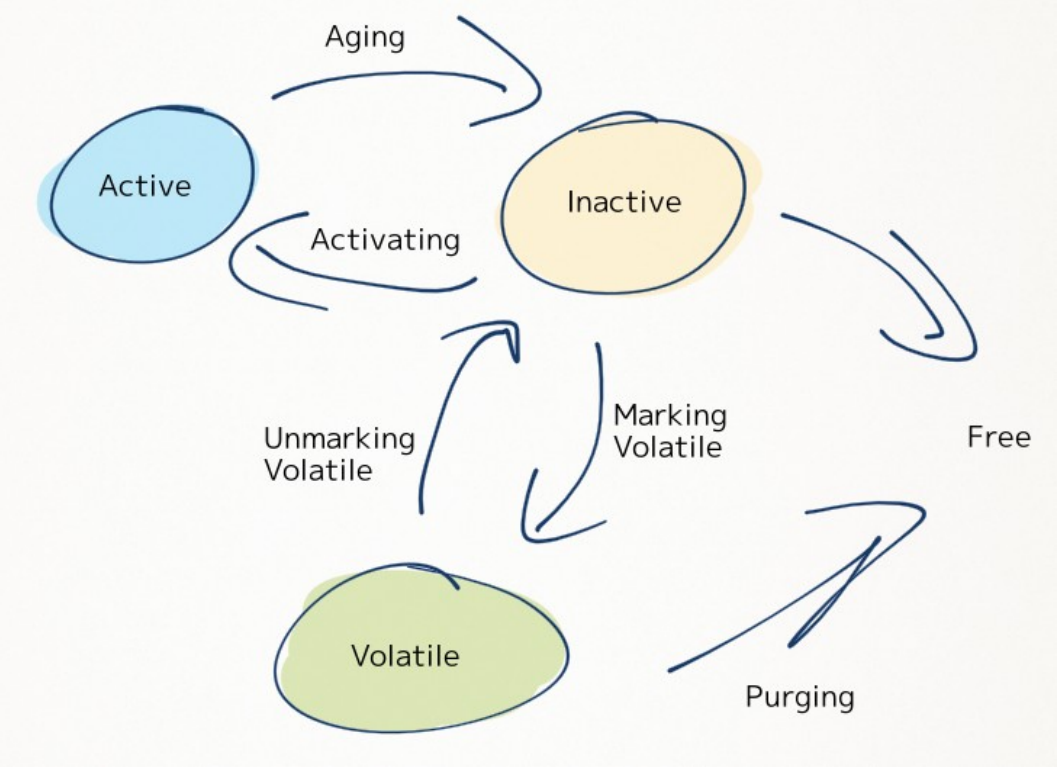
- Web Browsers
  - Off-screen data (images, tabs, etc)
- Memcached
  - Simplify cache sizing on nodes that also have other applications running
- Java object/JIT caches
- In-kernel uses?

Volatile Ranges Difficulties

- Android's ashmem and early implementations used a simple slab shrinker to trigger volatile range purging
- Unfortunately slab shrinkers are not numa aware
  - Could be tight on memory in one node and not the other, and could end up purging data on the wrong node

Volatile Ranges Difficulties

- Reworked to push deeper into the core VM, adding a new LRU page list
  - Managing at page granularity is much more expensive



Both approaches are needed

- Volatile ranges won't work for all use cases
  - Android's low-memory-killer
  - Cases where application coordination is needed to release cache data
  - Where Marking/unmarking ranges is too expensive
- Not an either/or decision

Implementation Status

- Unfortunately both vmevents and volatile ranges have been stalled
- Had a number of iterations, with limited feedback
- Can't seem to get attention from core MM developers
- No clear consensus

Feedback/Discussion

- API thoughts?
  - fallocate(fd, FALLOC_FL_MARK_VOLATILE, start, len);
  - madvise(addr, len, MADV_VOLATILE);
  - Signal/trap on volatile access?
  - Vmevent vs cgroup thoughts?
- Where else might these approaches be useful?
- Has the war on slab shrinkers gone too far?
- EZRECLAIM
  - Combine volatile ranges w/ clean file pages?
  - Use EZRECLAIM levels as vmevent waterlevels?

More Info

- Userland-low-memory-killer-deamon:
  - http://thread.gmane.org/gmane.linux.kernel.mm/84302
  - Email: Anton Vorontsov <anton.vorontsov@linaro.org>
- Volatile ranges:
  - https://lwn.net/Articles/500382/
  - Email: John Stultz <john.stultz@linaro.org>
- Join the discussion on lkml!

Legal

This work represents the view of the authors and does not necessarily represent the view of IBM, or Linaro.

IBM is a registered trademark of International Business Machines Corporation in the United States and/or other countries.

Android is a trademark of Google Inc. Use of this trademark is subject to Google Permissions.

Other company, product, and service names may be trademarks or service marks of others.