Summary: Coordinated and Efficient Huge Page Management with Ingens

Keywords:

- TLBs, page tables, page walks and virtual memory.
- Extended/Nested page tables.
- Base, huge, file-cached pages.
- Access, dirty bits.
- Virtualization, hypervisors.
- Contiguity, latency, tail latency, memory bloat, fragmentation.
- Radix trees.

Problem Statement:

- RAM capacities have been increasing rapidly to meet current computing requirements. TLBs not so much.
- As a consequence hardware virtual address translation is becoming more expensive.
- Workaround: Use larger sized pages. 4KB pages - base pages, 2MB pages - huge pages. Huge pages reduce TLB pressure.
- But all's not well, using huge pages have their own disadvantages and hence base pages are also used along with them.
- Now the challenge is for the operating systems (and the like) to have a good memory management framework to manage usage of huge and base pages. This is where Ingens comes in.
- Since by using huge pages we may quickly run out of contiguity, Ingens considers contiguity as a first class resource and manages it transparently.

Motivation for huge page support:

- Current systems have page tables as deep as 4-levels and a TLB miss to resolve a base page heavily impacts performance.
- Extended/Nested page tables have additional level of indirections and translations cost blows up with increase in number of levels. Hence base page translation is now even costlier.

Linux huge page policy:

- Greedy and aggressive. Advantages and disadvantages??
- Does not use huge pages for file cached pages. Why??

FreeBSD huge page policy:

- Very conservative about using huge pages in comparison to Linux.
- Reserves 2MB memory region in page fault handler but no instant promotion.
- Promotion occurs only when all base pages of huge page region are allocated.
- Considers a huge page for file caching only if all the pages in huge page region are read only or all modified. Why??

<u>Problems arising from poor huge page management</u>:

- Page fault latency:
    - Linux must zero hugepages.
    - Memory compaction is required to generate contiguous memory.
    - Low latency is important in some applications like web services.
    - Asynchronous-only promotion leads to significant loss in speedup.
- Fragmentation:
    - Linux does not track memory utilization and allocates pages greedily increasing fragmentation issues
- Unfair Performance:
    - Linux does not fairly redistribute contiguity thus leading to unfair performance.
- Memory sharing vs. performance
    - Hypervisors detect and share memory pages from different virtual machines whose contents are identical.
    - If the duplicated contents of a base page is present in a huge page of another VM, Linux demotes the huge page to share the page. This impacts performance if the huge page is frequently accessed.

<u>Design</u>:

- Goals
    - Reduce latency, bloat.
    - Provide fairness guarantees.
    - Find reasonable tradeoffs between memory savings and performance.
- Basic Primitives
    - Utilization tracking.
    - Access frequency tracking.
    - Contiguity monitoring.
- Threads
    - Promote-kth.
    - Scan-kth.
- Data structures
    - Util radix trees.
    - Access bit vectors.
    - Per-process memory promotion metric.
    - High and normal priority lists.

<u>Util bitvector</u> records which base pages are used within a huge page region. Each bit in the 512 bit vector when set implies that the corresponding base page is in use.

<u>Access bitvector</u> records recent access history of a process to its pages. Stores access frequency information computed by Exponential moving average.

Promote-kth:

- Entry points - periodically wakes up or is signalled by page fault handler.
- Compacts memory periodically and if necessary.
- Promotes fair allocation by allocating huge pages to processes having higher values of promotion metric computed by Scan-kth.
- Maintains two priority lists high and normal.
- High list is global - promotion requests from page fault handler - utilization above threshold.
- Normal list - per application list - filled as Promote-kth scans the address space.
- Promotion is in priority order.
- Does not reserve memory regions for promotion but allocates a huge page region and then copies base pages into it and maps into virtual space.

Scan-kth:

- Periodically scans process' hardware access bits in its page table to maintain per-process access frequency information in 8-bit access bitvector.
- Clears access bit only if the page is not frequently accessed or with 20% probability.
- Profiles idle fraction of huge pages and updates per-process memory promotion metric.

How ingens deals with above problems:

- Latency
  - Decouples promotion decisions from huge page allocation mechanism.
  - Page fault handler never allocates a huge page synchronously but signals Promote-kth to do its work.
- Memory bloat
  - Page fault handler decides to promote a huge page region to huge page only if utilization in the said region is above a threshold.
  - Unlike linux policy of demoting a huge page instantly after calling free, Ingens defers huge page demotion till utilization falls below a certain threshold.
- Fragmentation
  - Promote-kth compacts memory but only moves infrequently used pages.
- Page sharing vs. performance
  - Unlike linux Ingens denies page sharing if the base page region to be shared is present in a huge page that is frequently accessed.
  - If the page is no longer shared, which generally occurs when there is a write to the page it needs to be split, Ingens again checks utilization to promote the page.
- Proportional Sharing
  - Monitors and distributes memory contiguity fairly among processes.

Check your understanding:

1. Explain the virtual address space translation costs in systems that use extended page tables for translation.
2. Consider the following alternatives to using huge pages in order to reduce TLB pressure:

a. Instead of caching mappings of virtual address to base address of huge page in TLB, cache the mapping of virtual address to base address of the last level page table.
b. Similar to page replacement policy in which infrequently accessed pages are swapped into disk memory to allocate new pages or retrieve back disk-cached pages, consider a policy followed by hardware in which it swaps infrequently used TLB mappings into memory to create space for new mappings or retrieve back old mappings stored in memory.

What are the disadvantages in each approach that make using hugepages a better choice?

3. Out of the problems discussed above regarding hugepages which of them apply only to OS, to hypervisor and to both?
4. Consider a system which is heavily fragmented externally. Does this imply that it is also fragmented internally? What about the other way around?
5. Huge page allocation (in Linux/KVM) can potentially be extremely unfair to virtual machines in an asynchronous promotion setting. Explain.
6. In calculation of exponential moving average, why do very low or high values of alpha give higher misprediction ratios?
7. Promote-kth excludes applications having fewer hugepages from normal priority list. Explain why.
8. Hardware support for finer-grain tracking access and dirty bits for huge pages would benefit Ingens. Explain how.