Virtual Memory II & TMO

Virtual Memory Basics

Recall

Virtual memory serves two main purposes : it provides memory isolation between processes and it also allows processes to have more memory than the total physical memory through swapping.

How isolation is provided :

- Each process has its own set of addresses that is mapped to a space in physical memory
- Processes cannot directly access physical memory, the MMU translates from virtual address to physical address

Swapping :

- External storage can be used as a secondary memory where unused data can be stored in order to free space in physical memory

Segmentation and Paging :

There are two main methods of implementing virtual memory in an OS : segmentation and paging.

- Segmentation splits physical memory into segments of varying size contiguous memory and each process can only access memory within its segments
- Segmentation is prone to external fragmentation contrary to paging
- Paging partitions physical memory into fixed size frames that are dynamically mapped to pages of virtual memory
- In modern OSes, paging is used more often than segmentation

Memory Wall

The memory wall refers to the gap between the relative performance of processors and memory.

There are two components to the memory wall :

- The speed wall refers to the increasing difference in the speed of DRAM compared to CPUs
- The capacity wall refers to the fact that desired DRAM capacity by applications has grown faster than DRAM size

To combat the capacity wall, why don't we enlarge the physical memory size ?

- Each motherboard only has a fixed number of DRAM slots for each CPU which limits the max size of memory we can have (~2TB max)
- It is also costly to install a lot of DRAM if it is going to be underutilized most of the time

How can we extend available memory ?

If memory pressure is high, the OS can offload/swap data on secondary memory However, this requires an IO operation which can be pretty slow although the performance depends on the type of storage device. If the storage device is a hard drive, which is a mechanical device, the latency is very high as the head (a mechanical part in the hard drive) has to be moved to read/write.

Another possibility in a cloud environment is to use the DRAM of another machine to extend the local DRAM. In that case, a local machine communicates with a remote memory machine via a NIC (Network Interface Card) and RDMA (Remote Direct Memory Access).

RDMA defines a set of primitives in order to access DRAM on a remote machine. It even allows the local machine to access the remote memory without involving any of the OSes and can even write directly into application memory without writing into the OS' data buffers.

The performance of remote memory is dependent on the speed of the NIC as it is the bottleneck for sending/receiving packets.

Mechanism of swapping

Applications do not handle the swapping of pages themselves since the OS handles that. It also makes more sense since the OS knows which pages are hot or cold and thus decides which pages to swap out in situations of memory pressure.

The OS uses page faults as a mechanism for swapping :

- When a page is not present in memory => a page fault is triggered
- In the process of handling the page fault, the OS can decide to swap pages

Swapping a page between local and remote DRAM

Let's say that an application running on a local machine has the next 3 instructions :

- Store V0
- Store V1
- Load V2

When executing the last instruction, a page fault will be triggered since VA2 is not in memory. The OS will know thanks to the page table entry that the wanted page is

in remote memory. Then V2 will be swapped in from remote memory and take the place of V0 (the victim page) which will be swapped out onto the remote memory. The entry in the TLB will also be updated since V0 was replaced by V2.

Policies of swapping

Multiple decisions need to be made when it comes to swapping

- What pages to swap out ? => The OS needs to select the victim pages to swap out
- How many pages to swap out ? => The OS needs to quantify the deficiency of the DRAM
- When to swap ? => The OS needs to decide when is the right time to swap pages

How to identify the pages to swap out

There are two types of pages : Application pages and OS pages.

- Application pages contain the data allocated by the application, the code segments and heap and stack regions
- OS pages either contain kernel memory objects or serve as a page cache, that is, pages that cache files that are on storage

From the OS' point of view, there are two types of application pages : Anonymous pages and File pages (page cache).

- Anonymous pages are allocated by the mmap(MAP_PRIVATE) syscall and populated by the page fault. They must be swapped out first if it is possible to reclaim them
- File pages (page cache) are created by file operations or the mmap syscall. If they are clean, they can be immediately evicted as they are already on storage, otherwise they need to be written out

As a policy, Linux uses LRU (Least Recently Used) to select victim pages.

Since clean file pages don't need to be written out, the linux kernel has historically been biased towards reclaiming file pages rather than anonymous pages. This was mostly the case because storage devices used to be mostly hard drives which have terrible write performance thus making swapping very expensive.

As a solution to this, we can split the LRU lists between anonymous pages and file pages and try to balance the size of each list.

How many pages to swap out

Linux uses a set of **watermarks** to detect memory pressure.

- **WMARK_MIN** is the minimum level of free memory that the system should maintain. When free memory falls below this level, the system aggressively tries to free up pages
- **WMARK_LOW** is a low level of free memory. When this level of free memory is reached, the kernel starts moderately reclaiming pages to avoid reaching critical levels
- **WMARK_HIGH** is the desired level of free memory. As long as the level of free memory is higher, no pages are reclaimed

When to swap out pages

The **watermarks** guide the OS in when to swap pages.

When the level of free memory is below the low watermark, the **kswapd** daemon start freeing out pages until the level of free memory is greater than the high watermark

Data centers

Memory usage patterns in data centers

Data centers pay extra "taxes" in using memory as they have more overhead due to software packages, profiling, logging, etc.

This tax is estimated to be around 12-13% of memory.

Different clients (Virtual Machines) present different memory usage patterns. For example, about half of a "Video" workload's memory will be file-backed whereas a "Web" workload's memory will be mostly anonymous pages.

Memory architecture in data centers

Data centers use tiered memory for memory offloading. Different memory products can be selected for tiered memory (DRAM, Compressed DRAM, Persistent Memory SSDs, NVMe SSDs). They have different price tags (NVMe being the cheapest and DRAM the most expensive) and diverse performance characteristics (Compressed DRAM has performance close to that of DRAM but is incompatible with certain workloads).

Data centers use all these different products in a coordinated way in order to improve their cost efficiency.

In a tiered memory system, hot pages are kept in fast memory and cold pages are kept in slower tiers of memory.

Data centers need a lot of memory to support applications and the memory needs of emerging applications (like AI models) are growing massively.

Data centers thus need to make smart decisions to provision memory.

Challenges in swapping pages in tiered memory

- Existing solutions like g-swap don't allow the use of more than one slow tier of memory
- Compressed DRAM is not efficient for some workloads to use (overhead of compression/decompression is greater than memory being saved, this depends on the data being compressed)
- Offline profiling of applications can't reflect the runtime memory situation which is why we need more runtime accurate information

TMO (Transparent Memory Offloading)

TMO is a system developped by Meta to improve the efficiency of its data centers.

TMO uses **PSI** (Pressure Stall Information) to overcome the skewness of Linux page reclamation that prioritize file pages over anonymous pages. TMO also uses a userspace agent called **Senpai**, which has an online memory pressure detection mechanism.

High level architecture

The workload of an application is in a userspace container which communicates with the kernel through syscalls or page faults.

Senpai decides the memory policy to enforce based on the memory pressure information and instruments cgroup in the kernel to perform reclaim operations.

PSI

Existing OS solutions mostly rely on event counters like major/minor page fault counts. These counters are however insufficient to accurately capture the situation of an application.

For example, high major page fault counts could indicate that a workload is in the initialization stage and not that there is high memory pressure.

Thus we need a new metric to determine memory pressure : PSI.

PSI is used to specify the amount of lost work due to the lack of a resource.

It is composed of 2 metrics : a **"some"** metric and a **""full""** metric.

- **""some""** metric : it gives the percentage of time in which at least one process within the domain is stalled waiting for a resource
- **""full""** metric : it gives the percentage of time in which all processes are delayed simultaneously

The **""some""** metric measures the impact of resource insufficiency in each individual VM. The **""full""** metric measures the unacceptable losses of productivity that require immediate remediation.

Determining the memory requirements for VMs is very challenging as applications don't know if their memory is overprovisioned and the size of heap allocations is only determined at runtime.

Senpai uses PSI to determine the memory requirements for VMs.

If the **some** metric is high, it means that a VM needs more memory to be allocated to it. If the **some** metric is low, it means that a VM has too much memory and that TMO can reclaim some pages from this VM.

Like said earlier, there are two types of pages to consider as victims to swap out : file pages and anonymous pages.

TMO uses a metric called **Refault distance**.

Refault distance is the number accesses to unique pages made since the last reference to the requested page.

TMO counts the number of evictions that a file page has suffered and also counts the number of page faults for that page.

If the number of evictions is inferior to the number of page faults, it means that more faults happen after evicting file pages. In that case, TMO tries to reclaim less file pages.

Evaluation

TMO can provide memory of savings of up to 20% depending on the workload.

TMO also provides savings on the memory tax in data centers : the memory tax can be up to 13 points lower than without TMO.