Transcendent memory: Re-inventing physical memory management in a virtualized environment

Dan Magenheimer, Chris Mason, Dave McCracken, and Kurt Hackel, Oracle Corporation.
{first.last}@oracle.com (all non-students)

Introduction. Our objective is ambitious: to reinvent how physical memory is managed on a virtualized server. A native (non-virtualized) OS applies many well-known techniques to utilize a fixed amount of physical memory effectively. A virtualized OS also utilizes a fixed amount of physical memory, yet these same techniques may be less effective in a virtualized environment, where other virtual machines (VMs) competing for the same resources may have more urgent physical memory demands, higher priorities, or complex QoS requirements. Mechanisms for optimally provisioning CPU and I/O resources continue to evolve, but memory resource management appears to have stagnated since Waldspurger's ESX Server work and Disco, which date back nearly a decade. While ballooning and transparent content-based page sharing provide some flexibility for physical memory allocation decisions, we wish to go much further; we wish to convert as much physical memory as possible into a ***renewable resource*** so that it can be effectively *time-shared*. To do this, we revoke ownership of all but the most essential physical memory from each individual VM and place it in a pool, which is owned by the virtual machine manager (VMM) and controlled by a new ***memory scheduler*** in the VMM. Since "virtual memory" has markedly different connotations, we coin the term ***transcendent memory*** to refer to this pool.

While some recent virtualization research projects have studiously avoided changes to the OS and instead collect memory utilization information through passive inference, we embrace a future where OS's proactively identify whether or not they are running in a virtualized environment and adapt accordingly, employing ***paravirtualization*** to improve performance and/or enable more effective resource sharing with competing VMs running on the same physical machine. Yet we are not so foolish as to assume OS's will be completely rewritten overnight and so strive to mimimize the necessary changes, both in the OS kernel and in the environment in which the system runs. So, for example, though part of our objective might be achieved in future OS's with page-granularity hot-plug memory, we rely on more commonly available mechanisms.

Creating the transcendent memory pool. First we reclaim all ***idle memory*** from all VMs and place it in the transcendent memory pool. To do this, we artificially create memory pressure in each VM using ***aggressive dynamic feedback-directed ballooning*** to cause each VM to evict all of what its OS's pageframe replacement algorithm (PFRA) deems to be its lowest priority pages, the pages which comprise a rough approximation of what is effectively the complement of its working set at any given point in time. As the working set changes dynamically, the balloon is inflated or deflated accordingly (but notably *not* instantaneously). The ballooning mechanism funnels all of these now excess pages to the VMM, which places them in the transcendent memory pool. But now what can the hypervisor do with these pages?

Utilizing the transcendent memory pool. Since no PFRA can accurately predict the future, memory pressure may cause an OS to evict some pages that are needed again immediately, which leads to thrashing. To mitigate this, we provide a carefully crafted paravirtualized interface so that a portion of the transcendent memory pool can be effectively but indirectly used by each domain, but still synchronously reclaimed and repurposed as needed by the VMM. We call this interface ***hcache*** (pronounced "aitch-cash") because it is effectively a shared ***h***ypervisor-managed page ***cache***. Hcache provides no persistence guarantees, thus it can only be used for clean pages. Its usage is straightforward: An *hcache_put(page,handle)* is performed whenever a page is evicted from the page cache; this copies the page from memory owned by the VM into the transcendent memory cache. Then later an *hcache_get(empty_page,handle)* call is made prior to reading a page into the file/buffer cache; if the *hcache_get()* is unsuccesful, the page must be, as before, read from disk. Note that since no persistence guarantees are made, available memory can be dynamically and frequently reallocated as needed either between VM-specific hcaches or for other non-hcache usage, according to demand and/or memory scheduler policy parameters.

Next, we introduce ***hswap***, which *does* provide a persistence guarantee and can thus be used for dirty pages but, like hcache, does not have a fixed size; any *hswap_put()* can be summarily rejected. Hswap, as its name implies, serves to ameliorate the situation where a VM's working set grows faster than dynamic ballooning can deliver pages by acting as a pre-cache for a swap device. Hswap policy can be specified so that only well-behaved VM's, i.e. those that have actively participated in ballooning and hcache'ing, will find hswap pages available to reduce the excessive costs of swapping. Pages placed by the VM into hswap are part of the VM's state and counted against the VM's memory footprint, but can only be "addressed" indirectly and are not persistent across VM reboot, thus intentionally hobbling broader hswap usage for general data storage.

Future work. We envision using the transcendent memory pool as a cache for read-only and clustered filesystems mounted and shared by multiple VMs, as shared memory for inter-VM communication, and possibly directly accessing the pool from enterprise applications. We also believe hcache and hswap activity will be helpful to guide research into adaptable memory scheduler policy. Finally, concepts such as deduplication, compression and ghost caches are orthogonal and can be applied to transcendent memory as easily as VM-owned memory. We eagerly solicit feedback and ideas.

Submiited for OSDI'08 WIP session - Please do not redistribute 11/16//2008