

Transcendent Memory on Xen

Dan Magenheimer, Oracle Corp.
Extended Abstract for Xen Summit, February 2009

Introduction

Virtualization provides significant cost savings by allowing otherwise underutilized resources to be time-shared between multiple virtual machines. While efficient techniques for optimizing CPU and I/O device utilization are widely implemented, time-sharing of physical memory is much more difficult. As a result, physical memory is increasingly becoming a bottleneck in virtualized systems, limiting the efficient consolidation of virtual machines onto physical machines.

To overcome this memory bottleneck, it is necessary to optimize, across time, the distribution of a fixed amount of machine memory among a maximal set of virtual machines (VMs). To do this, we need simply measure the current and future memory need or *working set size* of each running VM and periodically reclaim memory from those VMs that have an excess of memory and either distribute it to VMs that need more memory or use it to provision additional new VMs.

However, attempts to implement this objective encounter a number of difficulties:

- Working set is a poorly-defined concept, so the current working set size of a VM cannot be accurately measured and an approximation must be deduced from a collection of externally observed data and events
- The future working set size of a VM cannot be accurately predicted, must be estimated from trends in externally available data, and may change unpredictable, perhaps dramatically, at any instant
- Memory has inertia and so cannot be randomly taken away from a VM without potential loss of data
- The consequence of underprovisioning the memory of a VM may be severe: A significant non-linear performance impact on that VM (i.e. *thrashing*)
- The consequence of mispredicting the number of VMs that can be supported by the fixed total machine memory is also severe: A significant non-linear performance impact on some set of VMs -- and thus on the overall system (i.e. *host swapping*).

Most research and development projects in this problem space focus on working set estimation and prediction. The transfer of memory between VMs is generally considered a solved problem -- *ballooning* [1] is implemented on all major virtualization platforms -- but memory inertia imposes challenges that ballooning can only partially overcome. Most notably, ballooning depends on requests to a VM to donate memory, which may have a long latency especially for large requests.

Our approach, Transcendent Memory, is fundamentally different in that it assumes that working set estimation and prediction will never be sufficiently accurate and, instead, provides a mechanism for mitigating the impact of that inaccuracy. The result is a dramatic reduction in paging and thrashing in many workloads and memory configurations, and thus an increase in VM and system-wide performance. At the same time, Transcendent Memory contributes a novel way of multiplexing physical memory and so provides greater flexibility for highly-dynamic virtual environments.

Transcendent Memory Overview

Transcendent Memory (*tmem* for short) provides a new approach for improving the utilization of physical memory in a virtualized environment by claiming underutilized memory in a system and making it useable where it is most needed. From the perspective of an operating system, *tmem* appears to be a fast RAMdisk of indeterminate and varying size that is useful primarily when real RAM is in short supply and is accessible only via a copy-based, semantically quirky interface.

Tmem requires guest cooperation, thus paravirtualization changes or *enlightenments* are required in the guest operating system, though the changes are surprisingly small and non-invasive. *Tmem* is complementary to compression and/or page-sharing and in fact may provide opportunities to optimize it by isolating memory that can most benefit. And while *tmem* partially depends on some form of ballooning-like mechanism, unneeded memory is pooled and utilized in a way that minimizes memory inertia, thus making a significant fraction of main memory instantly available to accommodate rapidly changing needs

More formally, Transcendent Memory is both: (a) a collection of idle physical memory in a system and (b) an API for providing indirect access to that memory. A *tmem* host (such as in a hypervisor in a virtualized system) maintains and manages one or more *tmem* pools of physical memory. One or more *tmem* clients (such as a guest OS in a virtualized system) can access this memory only *indirectly* via a well-defined *tmem* API which imposes a carefully-crafted set of rules and restrictions. Through proper use of the *tmem* API, a *tmem* client may utilize a *tmem* pool as an extension to its memory, thus reducing disk I/O and improving performance. While the semantic restrictions may seem quirky to a single client, they provide system-wide benefits in the form of reducing memory inertia and increasing “time-sharing” flexibility.

Implementation on Xen

More information on tmem can be found in [2,3]. Here we focus on a brief description of the Xen implementation.

All tmem operations are implemented as a single hypercall, *do_tmem_op()*, which takes as its single parameter an address of a data structure of type *tmem_op_t*. The first element of *tmem_op_t* specifies the operation to be performed and the other elements are obtained from a union of data structures dependent on whether the operation is: (a) setting control parameters or performing a privileged global operation; (b) creating or destroying a pool; or (c) operating on a pool.

When a tmem pool is created by a guest, the pool is assigned certain attributes: A pool may contain pages that are persistent or ephemeral, and may be private or shared. These attributes imply certain semantics on the operations performed on the pool. (Due to space constraints, we will limit our discussion to the semantics of operations on a private-ephemeral pool.)

Once a pool has been created, the core operations performed are *tmem_put_page* and *tmem_get_page*, or simply *put* and *get*. A *put* copies a page of memory from a guest into the pool and associates it with a *handle* consisting of a 32-bit *pool_id*, a 64-bit *object_id* and a 32-bit *index*. (The three parts of the handle are selected by the guest and uniquely represent, for example, a filesystem identifier, an inode number, and a page index into that inode.) A *get* searches the pool for a match for *handle* and, if found, copies the associated page from the pool back to the guest and removes the page from the pool (implementing the semantics of an *exclusive cache* [4]).

To efficiently handle a large number of pages with a sparse distribution of handles, each pool is implemented as a hashed-list of objects; each object is created as needed and serves as the root of a *radix tree* of nodes. The leaf nodes of the radix tree point to page descriptors, which in turn point to pageframes containing the actual data. Page descriptors are also kept in two doubly-linked LFU lists: one private list for each domain, and one global list across all domains. When memory runs dry, page descriptors and pageframes are reclaimed from the tail of one of these lists, depending on policy control parameters.

Since tmem is still evolving, the current Xen implementation is heavily instrumented, counting data structures allocated, operation frequencies and elapsed cycle count for various operations. The code is currently also liberally sprinkled with assertions and data structure sentinels to assist with debugging future work.

Evaluation, Future Work and Conclusions

Some benchmark results for a private-ephemeral tmem pool, referred to as *hcache* (or, in Linux, *precache*), have been described in full in [2] and are graphically depicted in Figures 1 and 2; to summarize, some workloads show throughput increases of 50% and disk reads reduced by nearly 95%. We expect mixed benefits in other workloads: large memory guests and applications with large sequential datastreams may see little benefit, while guests in high density consolidations may see more improvement.

We have completed implementation of a Linux patch for a private-persistent pool, which provides a *preswap* area that reduces guest swapping in memory-limited environments. Measurements for this will be forthcoming. We are also investigating an implementation of a shared-ephemeral pool, which can serve as a shared *precache* for cluster nodes co-residing on the same physical machine. And we believe a shared-persistent pool may serve as an interesting foundation for inter-domain communication.

Control plane code is under development. Extensive measurements on real-world workloads will certainly uncover tuning opportunities. Timestamps will also be used to measure page lifetimes in order to drive further space efficiency techniques such as compression and content-based page sharing.

Finally, tmem explicitly exposes certain previously hidden OS memory management operations, such as page eviction, that may uncover new avenues of virtualization research, not only in collaborative memory management but perhaps also in areas such as VM introspection and improved virtual disk management.

In conclusion, Transcendent Memory provides a novel approach for efficiently multiplexing physical memory in a virtualized environment across a (possibly non-constant) set of VMs with time-varying memory demands. The semantics enforced by the API minimize *memory inertia* while providing valuable benefits to the clients of the API. And we are encouraged that the flexibility of the approach suggests other as-yet-unseen opportunities and benefits.

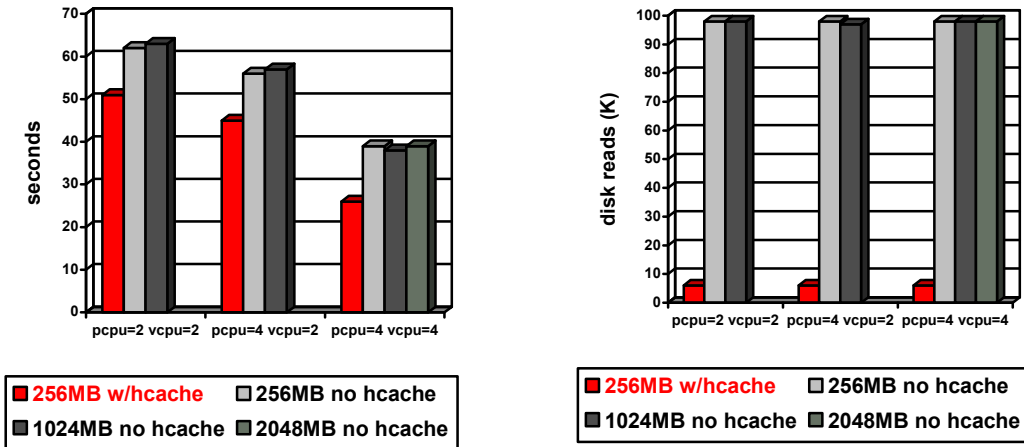


Figure 1. Benchmark: Linux compile, *cold* page cache, pre-caching *enabled* (ccache)

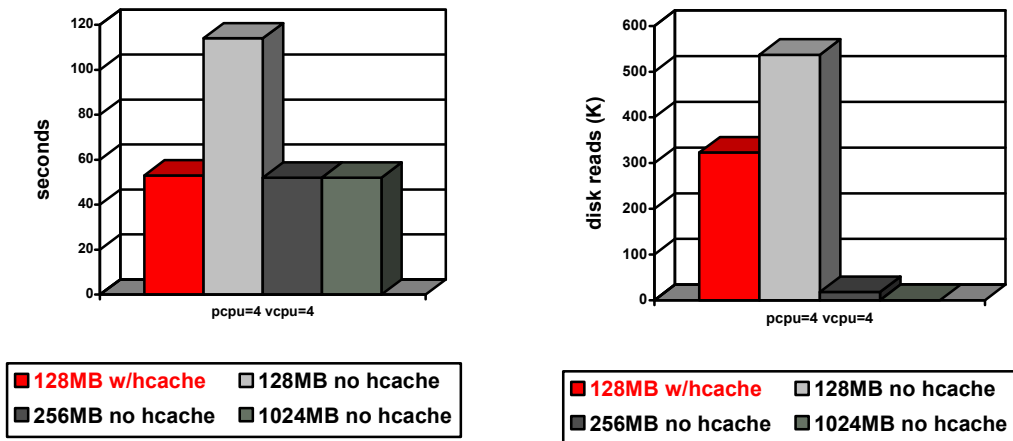


Figure 2: Benchmark: Linux compile, *warm* page cache, pre-caching *disabled*

References

1. C. Waldspurger. Memory Resource Management in VMware ESX Server, In Proceedings OSDI'02, <http://www.usenix.org/events/osdi02/tech/waldspurger/waldspurger.pdf>
2. D. Magenheimer, Paravirtualized Paging, In Proceedings WIOV'08, http://www.usenix.org/events/wiov08/tech/full_papers/magenheimer/magenheimer.pdf
3. Transcendent Memory Project, <http://oss.oracle.com/projects/tmem>
4. T.M. Wong and J. Wilkes, My Cache or Yours? Making Storage More Exclusive. In Proceedings Usenix ATC'02.