VMMem Services

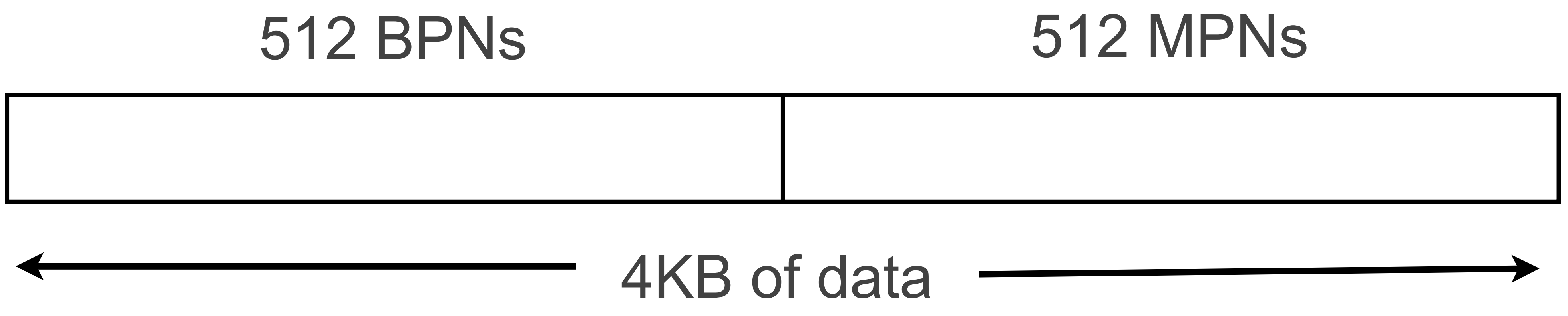
Alex Garthwaite
Monitor Group
December 18th, 2012

Confidential

Outline

- PageList structure
- High-priority services
- Periodic/low-priority services
- Ballooning

PageList Structure: new directions/64bit MPNs



- **Primary data structure for pages between vmm and platform**
  - status (success/fail VOID bit) encoded in 31st bit of BPNs
  - MPNs now only used for debug checks (vmm-platform matching) and remapping
- **Existing PRs**
  - eliminate MPNs from structure
  - stop using VOID bit and have BPNList and status
- **Other possibilities**
  - use single list at a time, not per-zap client
  - pass BPNs and statuses (or MPNs) separately
  - can stop populating MPNList during zap operation
- **Note: will need MPNList for pre/re-validation (e.g., remap)**
  - should evaluate benefits of ability to bulk-validate in vmm

New Framework for VmMem Services

- **Unify release/validation paths in the vmm**
- **Free platform scheduler**
  - move policy out of vmm
  - model: track and operate on not-exposed sets of pages (4K or 2M)
- **Address latency and overhead issues**
  - base work uniformly on guest progress
  - greatly reduce work in vcpu paths
  - track time stats to gauge latency bounds

Classes of Services

- High-priority
  - swap, p2mUpdate, page-retire
- Low-priority/best effort
  - NUMA-mig, 2M-defrag, page-sharing, ...
  - monitor/GPhys/frame mig, 2MB heuristics, page-checking
- Tension: hosted v. ESX (code & differences)

High Priority Cases

- Release operations (all invalidations):
  - swap: making progress wrt VMTrack
  - p2mUpdate: draining queue quickly/swap
  - page-retire: address failing HW
- Both polled and action-driven
  - responsive to allocation activity in vmm
- Do cases pending at start of check
  - vmotion, logging and action-starvation

High-Priority Services

VMM

```
while (PlatformGetRequest(&type, &numPages, &dataMPN)) {
    services[type].processRequest(numPages, dataMPN);
    PlatformCompleteRequest(numPages, dataMPN);
}
```

VMkernel

- Tracks pending requests on first call, defers ones arriving after
- Selects type and page-set
- Operates on pages once not exposed to VMM

vmware

Implementation

- **Primary source files**
  - VMM: vmcore/vmm/main/vmmem.c and vmcore/vmm/platform/*/platform.c
    - VMMemHandleReleaseRequests
  - Vmkernel: vmkernel/mem/vmmemservices.c
    - VmMemServices_PickupRequest, VmMemServices_CompleteRequest
  - Hosted: vmcore/vmx/main/vmmemServices.c
    - same as for ESX
- **Callbacks**
  - VMM: vmmServicesCallbacks[type]
    - currently only one per type: filtering function
    - different across services:
      - p2mUpdate: only shared pages
      - swap: avoid pages currently in translation
  - Platform: VMMEM_SERVICES_CALLBACKS, VMMEM_SERVICES_CALLBACK_FNS
    - selection, completion functions
- **Platform implicitly tracks first call for each type**
  - list length and PageList in dataMPN passed back and forth

Where we are and clean-ups

- **Selection and dataMPNs**
  - a given service may require multiple iterations/batches to complete
  - swap selects all pages up front (limits max swappable target)
  - p2mUpdate selects pages from its own queue
  - should just select as the exchange of each batch happens
- **Pending tasks at start of handling release-request**
  - 32-bit atomic field: pending boolean indicating work
  - platform maintains own pending bitmap, copy passed back and forth
  - could merge these and simplify the code
    - makes tracking aggregate stats, timing info. simpler
    - does bake in order services are evaluated
- **Need for better (timing) stats than currently exist in vmm**

Low-Priority Services

VMM

```
10HzCallback {
    init(&timeBound);
    while (PlatformGetRequest(&type, &numPages, &dataMPN, &timeBound)) {
        services[type].processRequest(numPages, dataMPN, &timeBound);
        PlatformCompleteRequest(numPages, dataMPN, &timeBound);
    }
}
```

VMkernel

- Prioritizes queue of requests based on timeBound
  - time since guest exit (ideal) or simple time-limit
  - E.g., sampling v. pshare v. remap-validation
- Selects type and page-set (avoid starvation)
- Operates on pages once not exposed to VMM
  - On-demand accesses by guest handled per page

Low-Priority Cases

- Invalidation
  - NUMA-migration, 2M defragmentation
  - page-sharing
  - 2M-region promotion
- Validation (needed? need perf evaluation)
  - NUMA-migration
  - checkpoint restoration
  - based on guest access patterns
- Sampling: possibly leave out

Low-Priority Cases

- Tracking why pages are not exposed
  - handle in page-fault path
  - alternative proposal: mix of in-line and async work
- Queues of work (and prioritization)
  - monitor-side operations
  - platform-side operations

Timing

- **Latency requirements**
  - memschedInfo.vmmem.prop.latencySensitivity: 64-bit value!
  - need map to timing constraints
- **Overall time**
  - time since guest exit too expensive
  - measure time since vcpu-progress stops as approx.
- **Current 10Hz period**
- **Per-type timing stats**
  - easiest to measure from vmm
  - useful in platform to choose work
  - ... but may not be issue: little in-line work

Prioritization

- **Monitor and Platform managed queues**
  - split time bound between two, interleave
- **Starvation concerns**
  - may not be an issue given new structure of services: work out-of-line
  - groups of types' pages can be batched
  - timing overall may be short: crosscalls/platform-calls dominate
- **Strawman proposal**
  - sampling > {migration, defrag, sharing} > {validations} > 2M-promotion

Interface

Initialization

- Configuration
  - latency limits
  - timebound for this round of processing
- Timing info
  - time exited guest
  - time this round started

Request

- Request state (op, etc.)
- Stats
  - time stats per operation type

Asynchronous Activity In Platform

- **Pages passed down not exposed**
  - need to allow page-faults before processing complete
  - may cancel operation on that page: swap-outs
  - may need to complete for those pages: page-sharing, remapping
  - requires knowing *why* page is being processed (easy)
- **Extensions**
  - with less impact on guest, re-evaluate page-sharing, NUMA-mig rates
    - base more on memory bandwidth
  - remapping at 2MB level
- **Accounting/scheduling asynchronous work**
  - basing on VM’s allotment may result in starvation of service
- **(Re)validation services may add overhead through buffering**

Ballooning

- **Measuring guest impact**
  - long-standing problem: measuring impact and backing off
  - dynamically changing boundary: not a static decision, how to handle this?
  - use guestLib: access to pinned info., etc. often easier from user-level in guest
  - learn from vmmon/hosted environment and improve *both*
- **Balloon targets and role of ballooning**
  - primarily constrains guest physical address space
  - secondarily releases memory (after guest execution to allocate pages)
  - so target may be much higher that perceived level of overcommit
  - allows some applications/guestOSs to constrain cache structures, etc.
  - what of guestOSs that maintain free-page pools (e.g., 10% of memsize in Windows)
- **Downward spiral (in general)**
  - previously: sampling bias (only tracked non-ballooned pages)
    - maintained wss percentages then applied to current non-ballooned size
  - guest less active, not on normal tasks when ballooning or swapping
    - impacts both pages touched and vcpu progress
  - what if free-page pools, etc. not all scaled down?
- **Fairness issue across VMs**
- **Will never solve problem of VMs too early in power-on cycle**

Recap

- **Userworlds/VMX (from yesterday)**
  - numa-migration support
  - backing with 2MB pages
  - better page-sharing?
- **High-priority services**
  - simplify both data (selection) and control (pending) aspects
  - add timing/stats
  - add page-retirement zaps (if not there yet)
  - bound VMX pinning (p2mUpdates)

Recap

- **Low-priority services**
  - redirect appBallooning queue to platform
  - services
    - split remap into two halves: invalidation, validation
    - evaluate need for (remap) validation (TPC-C runs on NUMA)
    - move page-sharing to new framework
    - consider moving 2M-defrag to this framework
    - consider prevalidation in vmm, “rearming” large regions
  - implement asynchronous platform code
    - support 2MB
    - dealing with page-faults on in-process pages
  - evaluate timing of services and how throttled
  - introduce (latency-aware) timing model
  - add better stats/timing info.
  - in vmm and platform, balance across pending queues to avoid starvation