The VMX*3 Project

Ariel Tamches

tamches@vmware.com

MonitorU, February 22, 2011

Outline

- **Introduction**
- Build Types
- VMX*3 How-To
- VMX*3 Details
- Conclusion

The Problem

- **Customer has an issue with a VM**
  - Escalated to support/GSS/CPD
- **But CPD not given tools they need**
  - Need beta/debug build if vmm or vmx went off into the weeds
  - Need stats-collecting build to gather performance data
    - Kstats, callstack, lockstats, etc.
- **In-house, this is not a problem (obj, opt build types)**
- **But in the wild, CPD is stuck diagnosing with a release build**
  - Which has almost no debugging; few performance measurements
  - One-off beta or opt builds? Time-consuming and not guaranteed to work
  - Need to engineer various vmm/vmx build types to run on a release build vmkernel

Solution

- **Starting in M/N: ship multiple build types of vmm & vmx**
  - Not just during beta program. GA too!
- **Allow users to switch build types, statically or dynamically**
  - Under CPD's guidance
  - In “tech support [shell] mode” so it won't get misused
- **VMX*3 = “release, beta, and stats”**
- **Investigations no longer hampered by build types that we ship to customers**
  - Frantic untested one-off builds no longer needed

Deja vu?

- **ESX Dali & 3.5: we shipped “vmx*2”: release and beta builds**
  - `debug=true` was used to select which to run
  - This was silently but intentionally removed in KL & KL.next due to replay incompatibilities
- **M/N: bringing it back to life**
  - Also add a performance-measurement build “stats” to make “vmx*3”
  - And tools to change between build types, at runtime or statically
  - New config option `vmx.buildType` = {`default`, `debug`, `stats`}

Outline

- Introduction
- **Build Types**
- VMX*3 How-To
- VMX*3 Details
- Conclusion

Build Types (vmm=vmx)

	Shipped to Customers in M/N			Internal Builds	
	release	**beta**	**stats**	**opt**	**obj**
VMX86_DEBUG		✓			✓
VMX86_STATS			✓	✓	✓
VMX86_DEVEL				✓	✓
License check?	✓	✓	✓		
VMX86_LOG					✓
optimization	-O2	-O2	-O2	-O2	-O1
Frame pointers		✓	✓	✓	✓

Beta / Debug Build: When?

- **If monitor goes off into the weeds and core file is bad**
  - A build with many internal assertion checks may fail earlier (core dump before VM went completely insane)
    - “Wear a fault on its sleeve”
  - May get a monitor crash before a guest BSOD (a good thing)
- **If the VM has deadlocked**
  - Lock rank checks only exist in debug builds
  - In non-debug builds, locks are allowed to deadlock
- **If frame pointers are needed**
  - For backtrace purposes, perhaps
  - Certain Vprobes scripts (gathering vmm backtraces)
- **(Maybe we should change beta to use -O1 for better core files?)**

Stats Build

- **When you want information about VMM (or its calls to VMK/VMX) and its overheads**
- **Runs almost as fast as a release build, and collects a wealth of performance data**
- **Stats – Counts of how often important code is reached**
  - Run $VMTREE/support/scripts/getStats.pl
- **Kstats**
  - VMM service times/counts
  - VMM Semaphore/Lock Stats
  - MX User Lock Stats
  - Crosscall Stats
- **Callstack - https://wiki.eng.vmware.com/CallstackProfiling**
  - VM CPU time in great detail

Kstats: The 1-Slide Version

- **Think of the monitor as a collection of services**
  - Exit HV, BT, or DE to perform a service (usually emulation) then back asap
- **Most kstats are instrumentation-based**
  - KSTATS_START – entering VMM to start a new service
  - KSTATS_PUSH/POP – stop previous kstat, start a new one
  - KSTATS_VECTOR – retroactively change the current kstat
  - Fast: no rdtsc on each push/pop to measure time
    - Time in a kstat is sample-based, 100/sec
    - Instrumentation only needs to set current kstat and count # of invocation
- **Other kstats are sampled-only**
  - BT, DE, HV – where even modest push/pop is too expensive
  - Just profile such kstats to get time (sacrifice getting # of invocations)
- **Howto: after a run, cd to “stats” subdirectory**
  - `$VMTREE/support/scripts/kstats.prl`

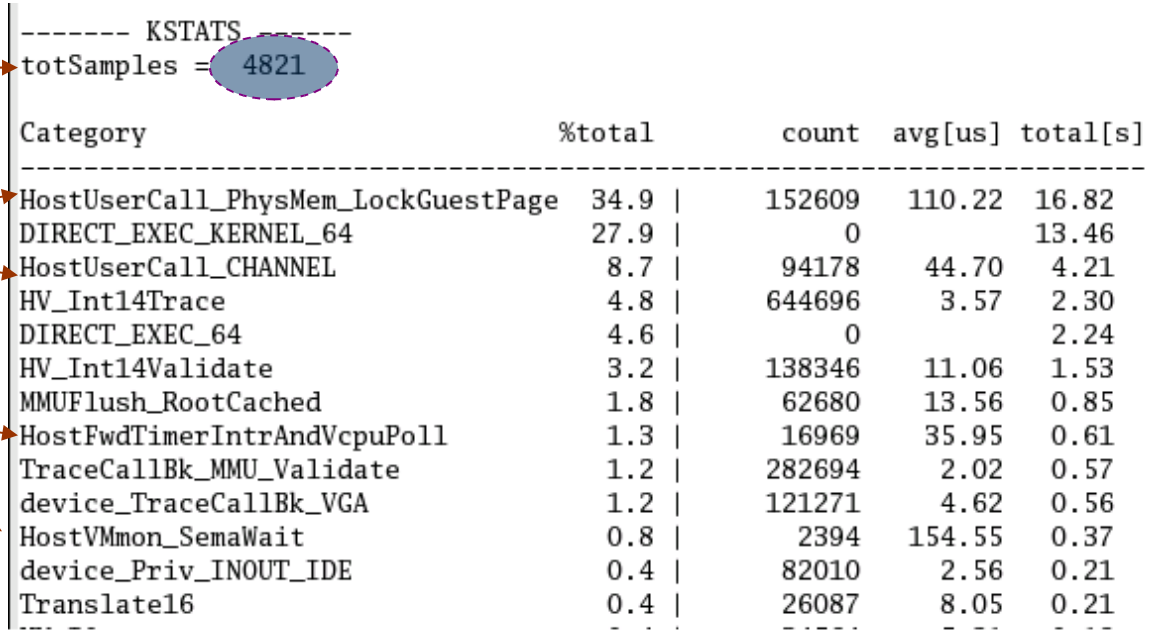
Kstats: Hosted Example

- **Includes UserRPCs to VMX, VMMon ioctls, VMKernel calls**
- **Includes all of a VM's elapsed time (not just CPU time)**
  - e.g. Including I/O wait, blocked on locks, vmkernel scheduler delays

 # samples

UserRPCs

VMmon calls



```
------- KSTATS -------
totSamples = 4821

Category                                %total        count  avg[us] total[s]
-------------------------------------------------------------------------------
HostUserCall_PhysMem_LockGuestPage       34.9 |       152609   110.22   16.82
DIRECT_EXEC_KERNEL_64                    27.9 |            0            13.46
HostUserCall_CHANNEL                      8.7 |        94178    44.70    4.21
HV_Int14Trace                             4.8 |       644696     3.57    2.30
DIRECT_EXEC_64                            4.6 |            0            2.24
HV_Int14Validate                          3.2 |       138346    11.06    1.53
MMUFlush_RootCached                       1.8 |        62680    13.56    0.85
HostFwdTimerIntrAndVcpuPoll               1.3 |        16969    35.95    0.61
TraceCallBk_MMU_Validate                  1.2 |       282694     2.02    0.57
device_TraceCallBk_VGA                    1.2 |       121271     4.62    0.56
HostVMmon_SemaWait                        0.8 |         2394   154.55    0.37
device_Priv_INOUT_IDE                     0.4 |        82010     2.56    0.21
Translate16                               0.4 |        26087     8.05    0.21
```

Kstats: If You Need More Detail, then What?

- **We see elapsed time in vmm & vmkernel, but...**
  - Is it CPU time, wait time, host I/O time, blocked on a lock, ...?
  - Kstats doesn't “drill down” (by design, kept simple)
- **We have more in our quiver**
  - kstats.prl prints more things:
    - VMM SemaphoreStats – cpu + blocking time in all vmm locks/semaphores
    - VMX Lock Stats https://wiki.eng.vmware.com/PIMXUserFacility
    - Crosscall stats (contact: kevinc)
  - Callstack
  - Vprobes (extensible custom instrumentation – contact: vprobes@vmware.com)

Callstack Profiler: Introduction

- **Where CPU time is spent *by call chain***
  - Knowing the call chain (as opposed to just a leaf function) is often helpful
  - A superset of flat profiling
  - Requires frame pointers
  - Very low overhead (< 1%)
  - Can view top-down (overview, then drill down to “hot” callees)
    Or view bottom-up (start at leaves, then examined “who called me”)
- **`cd stats; $VMTREE/support/scripts/vmmCallstack.pl`**
  - It'll gather vmm / vmkernel / vmx symbol tables and archive results
  - Follow onscreen directions
  - Run the helper script `./viewCallstack` that it generates
  - Run with `--help` for options

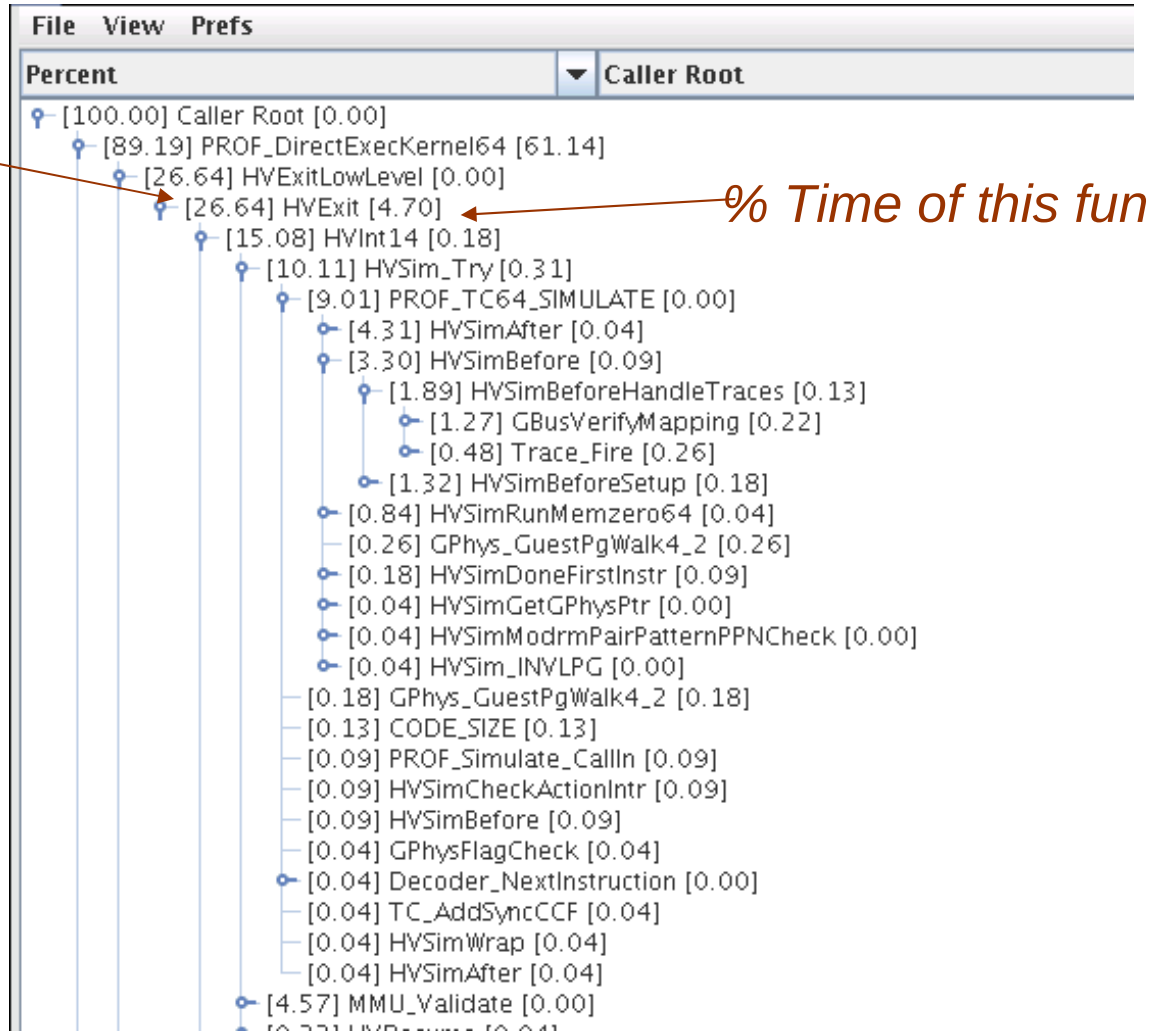
Callstack GUI: Hosted Example

% Time including callees

% Time of this function only

Click circles to Expand / unexpand callees

Functions can appear more than once – Multiple paths

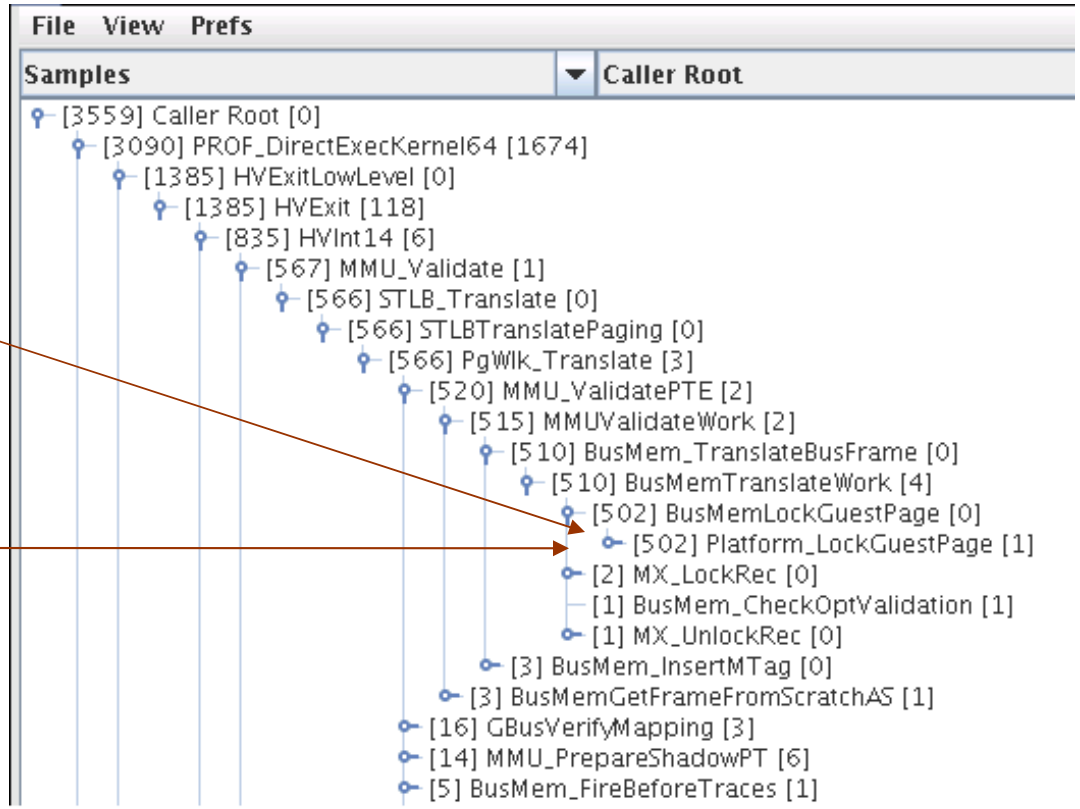


Callstack: Unified VMM/VMX/VMK Profiling

- **Can drill down through Hosted UserRPCs or ESX VMKCalls**

Kstat for Physmem LockPage (the cpu portion)

Can continue expanding down to vmx code

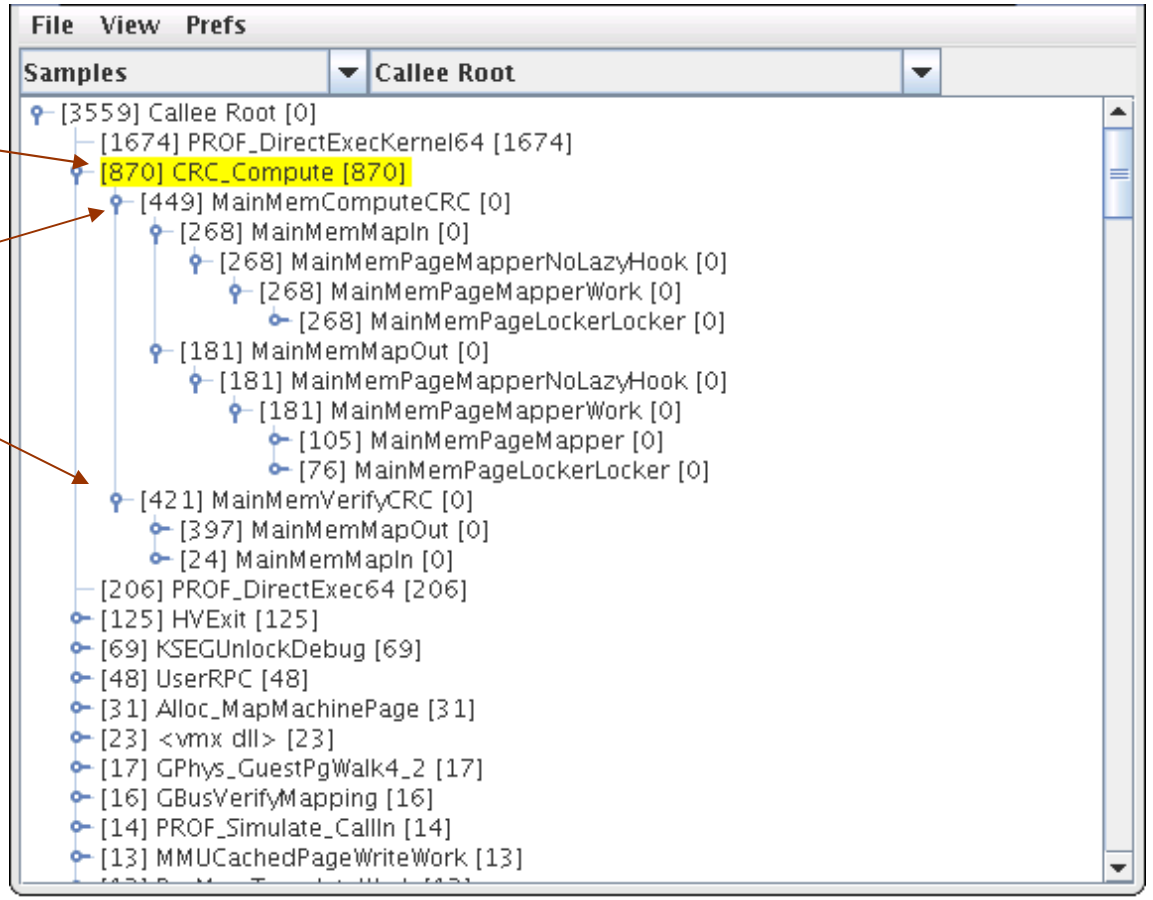


Callstack: Switching to Callee-Rooted View

- Now roots are leaf functions, children are its callers
- In this example, an XP Boot was optimized from 46 to 36 seconds

Leaf function

Who called it



We now have enough information to know what's going on and optimize

Other Callstack Features

- **Guest Kernel Profiling**
  - If user provides guest symbol table to vmmCallstack.pl, guest functions are shown in callstack GUI too!
  - [Guest -> VMM -> VMKernel] unified profile
  - Thorough breakdown of where CPU time is spent (except guest user-level)
- **Can use nmis as a periodic tick source**
  - `monitor.nmistats=1`
  - Measure time spent *in hardware* for HV exits and Interrupts!
  - Measure code where interrupts are disabled
  - Allows custom CPU events like cache misses
  - More info: https://wiki.eng.vmware.com/NmiProfiling
- **Coalescing time in a function**
  - `./viewCallstack --rootat` [funcname]
- **Instruction-grained profiling (“disassemble function”)**

Outline

- Introduction
- Build Types
- **VMX*3 How-To**
- VMX*3 Implementation
- Conclusion

VMX*3: How-To

- **Official for-CPD documentation is Internal KB #1030549**
  - "Using multiple VMX build types in ESX 5". Following is a clickable link:
  - http://knova-prod-kcc-vip.vmware.com:8080/contactcenter/php/search.do?cmd=
  - If that doesn't work go to http://knova-prod-kcc-vip.vmware.com:8080/contactcenter/microsites/microsite.do
- **Log onto an ESX system as root**
  - "Tech support mode"
- **Run: `/bin/vmx-buildtype --help`**
  - It is a python script that uses the vSphere API
  - Could copy & run it from any machine that has the API libraries
  - Low-level interface matches the intended use: under CPD's guidance
- **bora/support/scripts/vmx-buildtype is its perforce home**

vmx-buildtype: Common Options

- **Connecting:**
  - `--server localhost`
    `--server` [name of VC server]
  - `--host` [ESX machine name]
    - Only if --server pointed to a VC server
  - `--username` [name] `--password` [word]
    - To log into server
- **Identify a VM to change**
  - `--vmfile` [file]
    [file] matches “File” column of vim-cmd vmsvc/getallvms
  - `--vmname` [name]
    [name] matches “Name” column of vim-cmd vmsvc/getallvms
- **The new build type**
  - `--buildType` {`default`, `debug`, `stats`}

Examples

- **Change a VM to debug (beta) build, VM was powered on**
  - ```
    # vmx-buildtype --server localhost --vmname
    m0n0wall --buildType debug
    Suspending VM...
    Reconfiguring VM "[storage1] m0n0wall/m0n0wall.vmx"
    for vmx build type debug...
    Resuming VM... done.
    ```
- **Changing to a stats build, VM was powered off or suspended**
  - ```
    # vmx-buildtype --server localhost --vmfile
    "[storage1] m0n0wall/m0n0wall.vmx" --buildType
    stats
    Reconfiguring VM "[storage1] m0n0wall/m0n0wall.vmx"
    for vmx build type stats... done.
    ```

Getting Your Bearings

- **Query VM's current build type (no –buildType parameter)**
  - `# /bin/vmx-buildtype --server localhost --vmname m0n0wall`
    `VM [storage1] m0n0wall/m0n0wall.vmx uses vmx build type: default`
  - Or peek at “vmx.buildType” in .vmx file
- **A vmx*3 runtime transition runs a new vmx process**
  - New pid, rotated vmware.log (old one now at vmware-0.log), etc.
- **Tip: look at line 1 of vmware.log, vmx/vmm build type is there**
- **Tip: a stats build creates “stats” subdir in VM's runtime directory**
- **Remember to change back to default build type when you're done!**
- **vmx.buildType is per-VM. If the VM migrates, so will its build type**
  - vmotion to pre-M/N -> harmlessly ignored (always release build); re-applied when you vmotion back to an M/N+ system.

How It Works

- **If VM is powered-off or suspended**
  - vmx-buildtype issues vSphere API command to reconfigure vmodl: `Vim.Vm.FlagInfo.monitorType`
  - Which is tied to vmx.buildType
- **If VM is powered on**
  - Same, but brackets code with a VM suspend/resume
  - Beware cost on huge memory VMs – suspending *cannot* be cancelled!
- **Undocumented/untested fast suspend/resume**
  - `--fsr` option to vmx-buildtype
  - Untested in M/N – *think carefully* before using on a customer system
    - Will be the default is O/P
  - FSR like Storage VMotion & Hot-Add – finishes in ~3 seconds

Outline

- Introduction
- Build Types
- VMX*3 How-To
- **VMX*3 Details**
- Conclusion

Plumbing

- **Hostd launches vmx**
  - Its VMHS library picks one of `/bin/vmx`, `/bin/vmx-debug`, `/bin/vmx-stats`
  - Based on vmdb tied to vmx.buildType (& vmodl FlagInfo)
- **Quirk: For FSR, authd forks vmx but same principles**
- **Overhead Memory Issues**
  - VC is told that vmx/vmm overheadmem is independent of build type
- **VC Admission Control with FSR**
  - With FSR, there are temporarily 2 vmx processes
  - VC increases resource pool when vmx.buildType is changed
- **Both vmx and vmkernel build types are in vmware.log**
  - Useful for post-processing tools

Coding Landmines To Watch For!

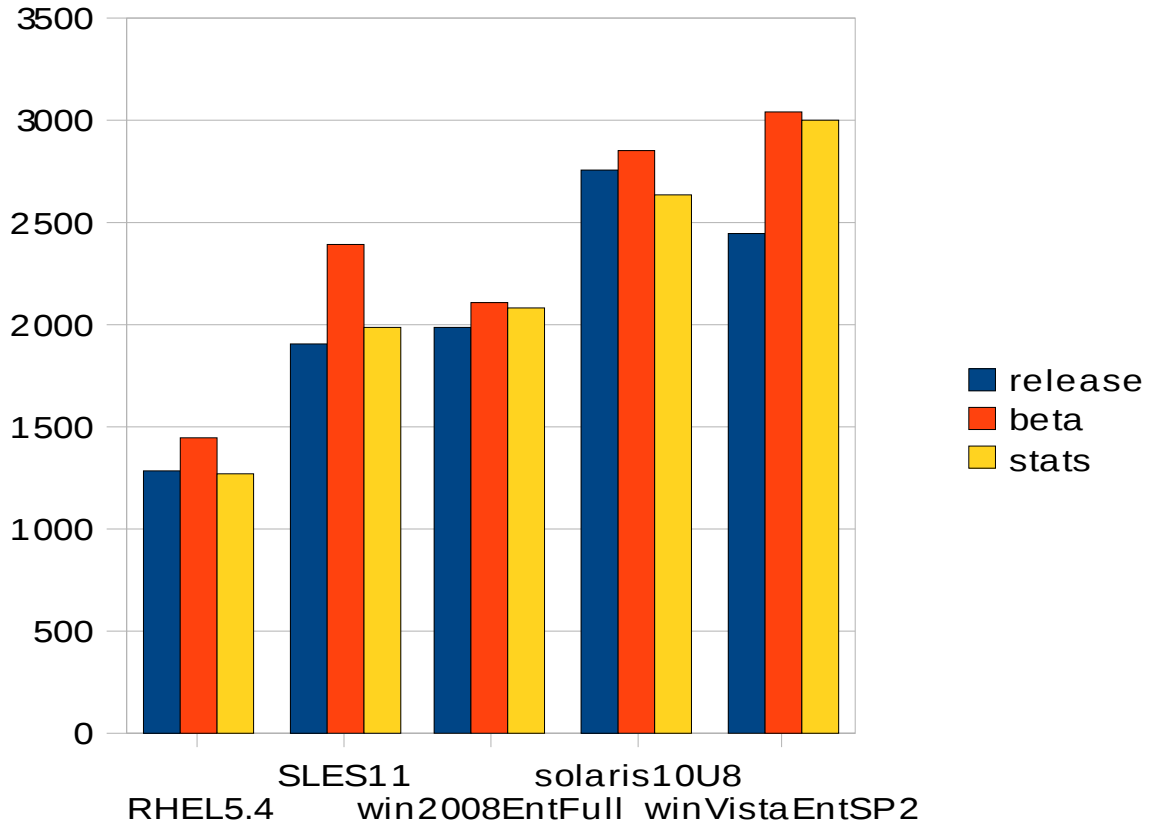
- **Remember that vmx/vmm build type may differ from vmkernel's**
  - Shared Area structs must be independent of build type
  - Some shared area variables may not exist – SharedArea_Lookup allowed to return NULL
  - VMM & VMX build types always same
- **vmx86_debug, vmx86_stats etc. only for your own build type**
  - Getting VMM/VMX build type from VMK: world->vmm->vmmBuildType
    - Example: Adding to kstats wait time after vmkernel wait loop
  - Getting VMK build type from VMX: GetVMKBuildType() syscall
- **Don't use VSI from VMX (has a brutal checksum that will kill vmx)**
- **Cannot use replay-based FT with vmx*3**

Testing VMX*3

- **test-esx script vmx-stats/vmx3.sh**
  - Creates several Vms, changes build types, asserts new vmx is launched, checks line 1 of vmware.log, etc.
- **frobos --vmx switch**
  - e.g. --vmx /bin/vmx-stats when running on a release vmkernel
- **Autoinstalls has a Build Type menu**
  - Random, default, release, debug, stats

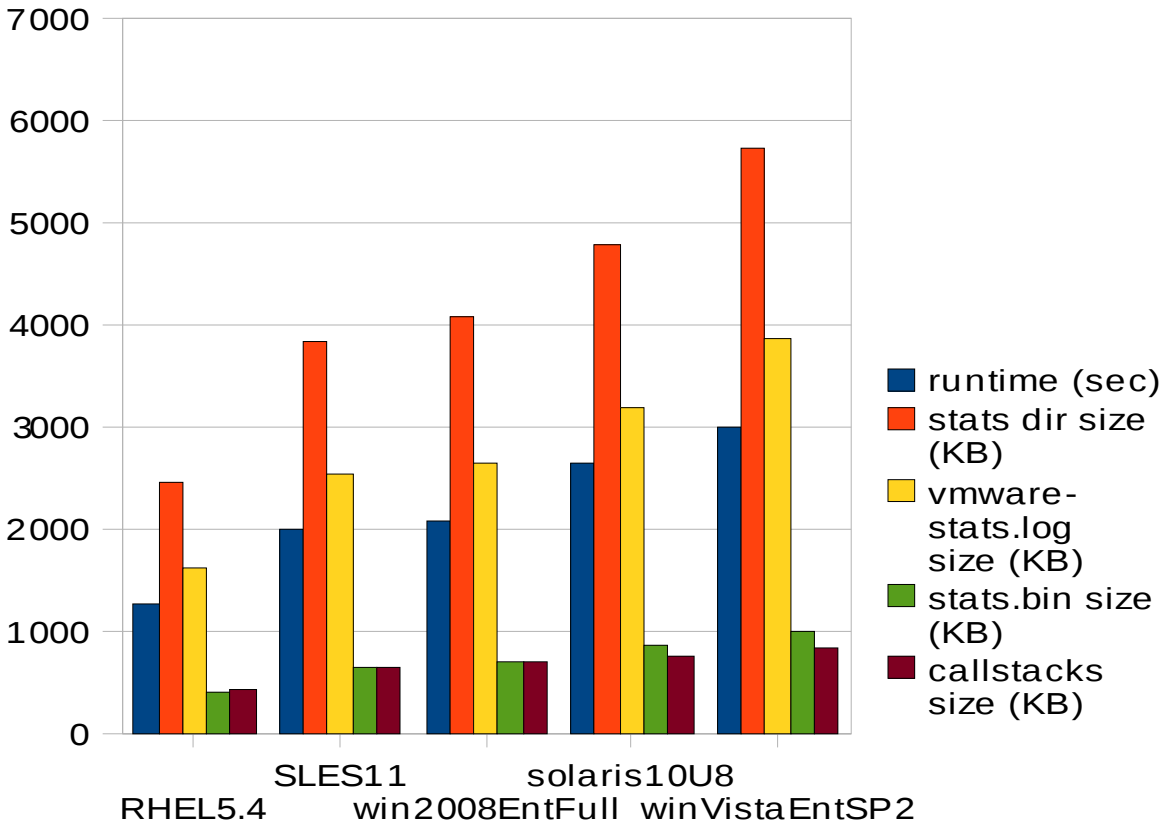
What's the Runtime Overhead?

- **A few examples from autoinstalls 2 VCPUs M/N (release vmkernel)**
- **Beta uses *much* more CPU; stats uses more disk**
  - Examples are best-case for beta, worst-case for stats (disk intensive)



How Big is the `stats` Directory?

- **Most stats overhead is disk, not cpu**
- **6 MB/hour isn't too bad but room for improvement**
  - This is with 2 vcpus. Some stats logging scales with vcpus.



Fine-Tuning Stats Overheads

- stats.enable=0 will turn off stats, kstats, semaphore stats, ustats
- monitor.callstack=0 will turn off callstack
- vmx.MXUser.statsEnabled=0 will turn off VMX's lock stats
- ustats.interval=[usecs] (default=10 seconds)
- vmx.callstackMaxFootprintMB=[n] (default=32)

Outline

- Introduction
- Build Types
- VMX*3 How-To
- VMX*3 Details
- **Conclusion**

Future Work

- **FSR for O/P**
- **VI Client GUI Integration**
- **Fusion / Workstation on hosted11 branch**
  - Workstation may get hostd – API should work fine
  - Fusion has already expressed interest
- **Vprobes integration**
  - Automatically switch to a stats build with FSR
  - Allows release build to remain lean; provide vmm frame pointers
- **Callstack improvements**
  - Further overheadmem reduction (shared trie in vmx)
  - No more vmmCallstack.pl after a run (error-prone for developers)
  - Automatically gather guest symbol table (dcovelli)
- **Log less in stats builds**

Summary

- **Starting in M/N, ESX ships with release, beta, and stats VMXs**
  - Use beta build for extra debugging/assertion checks
  - Use stats build to collect VM-centric performance data
- **The intended audience is CPD and Developers**
- **/bin/vmx-buildtype is the interface to change build type**
- **Beta has very large CPU overhead (well known)**
- **Stats has disk overhead**
  - Much faster than beta for a CPU-intensive workload, but barely faster for a disk-intensive workload.
- **Questions? Suggestions?**
  - tamches@vmware.com

Acknowledgements (incomplete!)

- **Development**
  - Carlos Robles (build, staging)
  - Taylor Hutt (replay structures)
  - Kevin Christopher, Jesse Pool, Doug Covelli (overheadmem)
  - Sriya Santhanam (hostd, vmodl)
  - Vmkernel team (test-esx, VSI)
  - Valeriy Zhuralev, Petr (authd)
  - Regis Duschesne (Fusion), James Lin (Workstation)
- **QE**
  - Divya Ranganathan, Rajesh Somasundaran (Monitor QE)
  - Jinto Anthony (System Test)
  - Saji Jacob (autoinstalls vmx*3 support)