A Tale of 10 Bugs: Performance Engineering at VMware

Ravi Soundararajan, SB '92 (VMware, Inc.)

ravi@vmware.com

MIT Guest Lecture, 6.172

11/18/14
The Key to Good Performance

“Make the common case fast…

…but make sure it is correct…

…and make sure uncommon cases are correct, too…”

(By the way, make sure it really is the common case)
Good Examples

Clean interface, fast answers
Common case:
User-level code
Another Example

Most Likely Destination
(if you need the map)
Missing the Common Case

- Complex
- Doesn’t leverage DC as likely destination
Where Am I?

Help hard to find
Where did my simple XP dialogue box go?
Cardinality?

- Common case is 1:1
- Why not an advanced option?
Why is this tough to do?

A convoluted common case example
A Performance Problem

2 Changes in VDI setup

1. Upgraded vCenter
2. Added a few new hosts

Suddenly, getting desktop (VM) is slow
Initial Analysis

Symptom: High CPU usage on vCenter

Why?

vCenter processing updates from vSphere hosts

(Observation: fewer updates in newer hosts → Virtualization HW support)

Updates ultimately cause license check → high CPU
Licensing: a red-herring?

Why is licensing expensive?
Usually not, but miss in vCenter cache ➔ expensive string comparison

Weird...License checks should not miss in vCenter cache 😞
Cache Misses?

Why the sudden license misses?

- Added hosts caused vCenter cache overflow
- But...vCenter cache *much* bigger than previous release
Resolution

Good: we anticipated cache increase in vCenter
Bad: Bug in upgrade meant OLD cache size was used 😞
3 big customers were impacted in the same week
Common Case Scorecard

Make the common case fast?
  Yes: Cache prevents expensive license checks

Make sure it is correct?
  Yes

Make sure it is the common case?
  Yes: License checks are the common case. BUT WHY????????

Make the uncommon cases correct?
  No! Upgrade uncommon and wrong
Why did I show you this example?

Illustrates Complexity of Products and Debugging
   Touches entire stack from VMM all the way to VDI

Highlights Scalability
   Problem exacerbated by adding more hosts

Interesting Plot Twist: what HW do you design for?
   Problem may not have occurred if all hosts were new
Remainder of talk

9 more bugs

• Some annoying (networking)
• Some about languages (Java, C)
• Some about platforms (Linux, Windows)
• Some about hypervisors (CPU/Memory issues)
The Right Tool for the Job
A Simple Networking Performance Problem
Networking and ssh (1/4)

Basic problem: ssh is slow

loginSshSlow.avi

20s from connection attempt to asking for password

Why?
Networking and ssh (2/4)

Verbose logging on server and client

Client:
```bash
ssh -vvv root@10.135.193.1 -p 1026
```

Server (n.b., my ssd running on 1026):
```bash
/usr/sbin/sshd -p 1026 -ddd
```

`loginSshSlowWithVerboseServer.avi`

(avi file shows just verbose server logging)

Seems to be a server-side issue (duh!)
Networking and ssh (3/4)

A useful tool: strace ➔ Traces system calls

% strace -tt /usr/sbin/sshd -p 1026 -ddd

01:20:50.069828 stat("/etc/resolv.conf", {st_mode=S_IFREG|0644, st_size=347, ...}) = 0
01:20:50.069915 open("/etc/resolv.conf", O_RDONLY|O_CLOEXEC) = 4

... 

01:20:50.070167 read(4, "# Dynamic resolv.conf(5) file fo"..., 4096) = 347

... 

01:20:50.070682 connect(4, {sa_family=AF_INET, sin_port=htons(53), sin_addr=inet_addr("10.0.2.3")}, 16) = 0

... 

REVERSE DNS LOOKUP

01:20:50.070947 poll([fd=4, events=POLLIN], 1, 5000) = 0 (Timeout)

<!!! 5 SECOND GAP !!!>

01:20:55.076323 poll([fd=4, events=POLLOUT], 1, 0) = 1 ([fd=4, revents=POLLOUT])
Networking and ssh (4/4)

Problem:

• Reverse DNS lookup to wrong DNS server
• Two 5s timeouts before proceeding

2 solutions:

1. Ignore DNS in sshd (ok in lab, not production)
   
   % /usr/sbin/sshd -p 1026 -ddd -o "UseDNS no"

2. Fix DNS server setting (better!)

Trying #1 validated issue, and #2 fixed it for real

loginSshFast.avi
Java Memory Management
Java memory management is done by the Java virtual machine.

Garbage Collection: Find ‘unreachable objects’ and delete them.

Java Garbage Collection

“Mark, sweep, and compact” garbage collector:

• Mark: identify garbage
• Sweep: find garbage on heap, de-allocate it
• Compact: collect all live memory together

Java Memory (not including code cache)

Heap

OldGen: existed some time in survivor space
Survivor: survive GC of Eden
Eden: newly-created
PermGen: class definitions, etc.
Java GC and Tuning Notes

GC for Eden is frequent and hopefully low overhead.
GC for “Oldgen” is less frequent and more CPU-intensive than Eden.
Rule of thumb: most (80%?) of memory is short-lived.

Many tunables in Java:

- Heap sizes (-Xms, -Xmx)
- Desirable ‘free heap’ ratio
- Survivor-to-Eden ratio
- Type of GC (serial, concurrent, mark/sweep, etc.)
- Number of GC threads
- Stack size (thread stacks NOT part of heap memory)
- Permgen size (not part of heap)

Profiling tools

- Yourkit, VisualVM, JMX counters, etc.
Min heap = max heap

→ Less incentive to GC

OLDGEN

Eden

(Survivor negligible)

OldGen Growth: 5GB to 20GB!

Few OldGen GCs.

Why?

Min = max? Usually good only if you know what you need.
Fixing the pathology

--Shrink max heap setting

--Do not set Xms (initial heap). Do not set initial Permgen

4GB
Oldgen
(vs. 20GB)
More
Frequent
GCs.
Another example with min heap = max heap

Min heap = max heap (bad)

Xms=Xmx=695m

Eden generation
Frequent GC
ok
Survivor generation
Old generation:
Growing = may be bad
Non-heap
(permgen, etc.)
Fixing the JVM settings: no permgen, no min heap

Lower max heap setting

Do not set min heap and do not set permgen: overall mem goes from ~400MB ➔ ~150MB
CPU profiling and diamond patterns
32-bit vs. 64-bit (Thanks, R. M.!!)

Benchmark run
  • Build A: 100 ops/min.
  • Build B: 50 ops/min.

What was the difference?
  • Build A: 32-bit executable on 64-bit hardware
  • Build B: 64-bit executable on 64-bit hardware

Huh?
CPU Saturation in 64-bit case

CPU is mostly saturated (in 32-bit case, CPU is not saturated)

CPU Saturated → GOOD USE CASE FOR SAMPLING PROFILER
What _is_ xPerf?

Runs on Windows 2008

Sampling profiler (with other cool attributes)

Records stack traces

Give caller/callee information
Look at Sampling Profile

Shows stacks originating from root
Shows 87% CPU used from 1 process
But this is just the thread start routine, where threads originate
The Perils of Sampling Profilers

From Root, most of the samples are from this call stack. Most popular stack, but is this the problem?
Perils of Sampling Profilers, Part 2

Most-common trace: not necessarily where time is spent

Many paths to “Tiny Function”
Maybe time spent here?
The Caller View

Look at Callers for various routines in stacks

<table>
<thead>
<tr>
<th>Callers</th>
<th>Weight</th>
<th>% Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>ntdll.dll!ZwQueryVirtualMemory</td>
<td>3,123,003.752...</td>
<td>77.26%</td>
</tr>
<tr>
<td></td>
<td>- ntdll.dll! ?? ::FNODOBFM:: `string'</td>
<td>3,109,241.648...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MSVCR80.DLL!_RTDynamicCast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MSVCR80.DLL!_RTtypeid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntdll.dll!ZwQueryVirtualMemory</td>
<td>2.004 444</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Not called a lot from root, however…
Called from few places and takes 77% CPU!
RTtypeid?
**RTtypeid?**

<table>
<thead>
<tr>
<th>Callers</th>
<th>Weight</th>
<th>% Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSVCR80.DLL!_RTtypeid</td>
<td>1,583,683,660</td>
<td>39.18</td>
</tr>
<tr>
<td></td>
<td>- vmacore.dll!Vmacore::ObjectImpl::IncRef</td>
<td>828,866,361</td>
</tr>
<tr>
<td></td>
<td>- vmacore.dll!Vmacore::ObjectImpl::DecRef</td>
<td>725,473,308</td>
</tr>
<tr>
<td></td>
<td>- MSVCR80.DLL!_RTDynamicCast</td>
<td>28,925,004</td>
</tr>
<tr>
<td></td>
<td>- vpxd.exe!ManagedObjectMapper::operator()</td>
<td>283,942</td>
</tr>
<tr>
<td></td>
<td>- [Root]</td>
<td>105,035</td>
</tr>
<tr>
<td></td>
<td>- vpxd.exe!VpxLRO::GetStatsContext</td>
<td>18,005</td>
</tr>
<tr>
<td></td>
<td>- vpxd.exe!DrmModule::SnapshotDomain</td>
<td>10,999</td>
</tr>
<tr>
<td></td>
<td>- vmacore.dll!Vmacore::PrintFormatter::FormatException</td>
<td>1,002</td>
</tr>
</tbody>
</table>

Hmm. RTtypeid is used in figuring out C++ type 39% of overall CPU? IncRef and DecRef are main callers.
void ObjectImpl::IncRef()
{
    if (_refCount.ReadInc() == 0) {
        const type_info& tinfo = typeid(*this);
        FirstIncRef(tinfo);
    }
    ...
}

typeid(): needs run-time type info (RTTI)
RTTI has pointers in it
But Why is 64-bit slower than 32-bit?

Runtime type info (RTTI) has a bunch of pointers

• 32-bit: pointers are raw 32-bit pointers

• 64-bit
  ▪ Pointers are 32-bit offsets
  ▪ Offsets must be added to base addr of DLL/EXE in which RTTI resides
  ▪ Result is a true 64-bit pointer

But wait...why is addition slow?
Why Is Addition Slow? Well, it isn’t…

Addition isn’t slow, but…

Determining module base address can be slow

• To find base address, RTtypeid calls RtlPcToFileHeader
• RtlPcToFileHeader grabs loader lock, walks list of loaded modules to find RTTI data
• This can be slow
• N.B.: This is why we see calls to zwQueryVirtualMemory

For more info:
http://blogs.msdn.com/junfeng/archive/2006/10/17/dynamic-cast-is-slow-in-x64.aspx
What Did We Learn?

RtTypeld is called from a bunch of places

RtTypeld is not, however, called from Root too often

RtTypeld is small and fast: not main contributor in most stacks (except IncRef and DecRef)

Lots of little calls add up

Caller view was important here!

(btw: 2 solutions:

• 1. Statically compute base addr and cache

• 2. Use latest runtime library, which avoids RtlToPcFileHeader)
Of course, maybe we should reconsider design

Do we need multiple inheritance and *dynamic_cast*?

```cpp
class D : public B, public C {
public:
    virtual ~D();
    virtual void foo();
};
```
Multiple inheritance and dynamic_casts

```cpp
class B : public A {
public:
    virtual ~B();
    virtual void foo();
};

class C : public A {
public:
    virtual ~C();
    virtual void foo();
};

class D : public B, public C {
public:
    virtual ~D();
    virtual void foo();
};

D* ptrD = dynamic_cast<D*>(ptrA);
```

Why do we use multiple inheritance?

• Store data as Object *
• Upon retrieval, do dynamic_cast
• Many objects need to inherit from various parents

Mallocs, Strings, and Ints
Microbenchmarks and macro conclusions
Question: How efficient is your software?

VMware software spans many layers:

• Virtual Machine monitor
  ▪ Needs small footprint for best performance
  ▪ Any CPU cost becomes virtualization overhead: slower guests

• Kernel

• Higher-level application software

⇒ For best performance, apply ‘monitor’ techniques to higher-level software
RDTSC: read timestamp counter

Lets you see the number of cycles for a section of code

```c
#if defined(__x86_64__)

static __inline__ unsigned long long rdtsc(void)
{
    unsigned hi, lo;

    __asm__ __volatile__ ("rdtsc" : "=a"(lo), "=d"(hi));
    return ( (unsigned long long)lo) | ( ((unsigned long long)hi)<<32 );
}

#endif
```
Rdtsc malloc/free test

```c
long t1 = rdtsc();
for (long i = 0; i < num_iters; i++) {
    testFoo = (foo_t *)malloc(sizeof(foo_t));
    free(testFoo);
}
long t2 = rdtsc();

printf("malloc/free loop 1st time average latency: %llu \n", (t2-t1)/num_iters);
```

➤ On average, about 50 cycles per malloc, 50 cycles per free
➤ Variance: occasional memory issues ➤ 500 cycles per iteration
➤ Is 50 cycles per malloc ok for you?
static void *
_int_malloc (mstate av, size_t bytes)
{
  INTERNAL_SIZE_T nb;               /* normalized request size */
  unsigned int idx;                 /* associated bin index */
  mbinptr bin;                      /* associated bin */
  mchunkptr victim;                /* inspected/selected chunk */
  INTERNAL_SIZE_T size;             /* its size */
  int victim_index;                 /* its bin index */
  mchunkptr remainder;              /* remainder from a split */
  unsigned long remainder_size;     /* its size */
  unsigned int block;               /* bit map traverser */

  ...
Malloc from glibc (2/2)

`/*

Convert request size to internal form by adding SIZE_SZ bytes overhead plus possibly more to obtain necessary alignment and/or to obtain a size of at least MINSIZE, the smallest allocatable size. Also, checked_request2size traps (returning 0) request sizes that are so large that they wrap around zero and aren't aligned.

*/

checked_request2size (bytes, nb);

... (lots more code) ...

The point is that malloc isn’t free.

Other options: Different malloc libraries? Custom memory management?

Lots More Code 😊
Rdtsc string vs. integer compare

```c
int num_iters;
char *s1, *s2;

int t1 = rdtsc();
for (int i = 0; i < num_iters; i++) {
    int equal = strncmp(s1, s2, strlen(s1));
}
int t2 = rdtsc();

int t1 = rdtsc();
for (int i = 0; i < num_iters; i++) {
    int equal = (num1 == num2);
}
int t2 = rdtsc();
```

String comparison: 81 cycles per loop

Integer comparison: 6 cycles per loop
### Strncmp: 81 cycles

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>400aa7:</td>
<td>48 8b 45 50</td>
<td>mov 0xfffffffffffffff0(%%rbp),%rax</td>
<td></td>
</tr>
<tr>
<td>400aab:</td>
<td>48 c7 c1 ff ff ff ff</td>
<td>mov $0xfffffffffffffff,%%rcx</td>
<td></td>
</tr>
<tr>
<td>400ab2:</td>
<td>48 89 85 20 ff ff ff</td>
<td>mov %rax,0xfffffffffffffff20(%%rbp)</td>
<td></td>
</tr>
<tr>
<td>400ab9:</td>
<td>b8 00 00 00 00</td>
<td>mov $0x0,%eax</td>
<td></td>
</tr>
<tr>
<td>400abe:</td>
<td>fc</td>
<td>cld</td>
<td></td>
</tr>
<tr>
<td>400abf:</td>
<td>48 8b bd 20 ff ff ff</td>
<td>mov 0xfffffffffffffff20(%%rbp),%rdi</td>
<td></td>
</tr>
<tr>
<td>400ac6:</td>
<td>f2 ae</td>
<td>repnz scas %es:(%rdi),%al</td>
<td></td>
</tr>
<tr>
<td>400ac8:</td>
<td>48 89 c8</td>
<td>mov %rcx,%rax</td>
<td></td>
</tr>
<tr>
<td>400acb:</td>
<td>48 f7 d0</td>
<td>not %rax</td>
<td></td>
</tr>
<tr>
<td>400ace:</td>
<td>48 8d 50 ff</td>
<td>lea 0xfffffffffffffff(%%rax),%rdx</td>
<td></td>
</tr>
<tr>
<td>400ad2:</td>
<td>48 8d b5 50 ff ff ff</td>
<td>lea 0xfffffffffffffff50(%%rbp),%rsi</td>
<td></td>
</tr>
<tr>
<td>400ad9:</td>
<td>48 8b 7d 00</td>
<td>mov 0xfffffffffffffff0(%%rbp),%rdi</td>
<td></td>
</tr>
<tr>
<td>400add:</td>
<td>e8 76 fb ff ff</td>
<td>callq 400658 <a href="mailto:strncmp@plt">strncmp@plt</a></td>
<td></td>
</tr>
<tr>
<td>400ae2:</td>
<td>89 45 bc</td>
<td>mov %eax,0xfffffffffffffffbc(%%rbp)</td>
<td></td>
</tr>
</tbody>
</table>
Integer compare: 6 cycles

```
    equal = (red_apple_six == inputNum);

400d8b:  8b 45 b0          mov    0xffffffffffffffffb0(%rbp),%eax
400d8e:  83 f8 06          cmp    $0x6,%eax
400d91:  0f 94 c0          sete   %al
400d94:  0f b6 c0          movzbl  %al,%eax
400d97:  89 45 ac          mov    %eax,0xfffffffffffffffffac(%rbp)
```

Straight-line code, no function calls.

👉 For performance, prefer ints over strings if possible
Strings and Things
Memory allocation differences between Linux and Windows
Memory differences: Linux vs. Windows

Motivation: runtime memory was 2x in Windows vs. Linux

Why?
Offsets in Windows (from windbg)

0:000> dt vpxd!VmMo -v

...+

+0x7b8 _configId : std::basic_string<...>
+0x7e0 _layoutId : std::basic_string<...>
+0x808 _layoutExId : std::basic_string<...>

...>

➡️ at least 40B between strings no matter what
Offsets in Linux (from gdb)

> gdb vpxd vpxd.core

(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_configId)
0x5f0

(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_layoutId)
0x5f8

(gdb) printf "0x%x\n", &((('VmMo' *) 0)->_layoutExId)
0x600

Only 8B between strings. Why?
Strings in Windows

```c
{
    std::__Container_base_12  // ptr 8B
    _Bx (union) {             // 16B
        _Buf                 // The string, if it fits
        _Ptr                 // ptr to string, if not
        _Alias
    }
    _Mysize  // 8B
    _Myres   // 8B (reserved space)
}

• Note: 40B minimum for each instance of the string
```
Windows Strings

Example from Visual Studio
Strings in Linux (from glibc documentation)

```c
_M_dataplus        # Default cost of any string
{
    _M_p            # 8B Ptr to char[] of string body
}

# String body ➔ 24B + sizeof(char [])
{
    _M_length       # 8B
    _M_capacity     # 8B
    _MRefCount      # 8B Reference Count
    char []         # the string (shared among instances)
}
```

Important: _MRefCount allows string body sharing!

20 instances of string: 19 are 8B, 1 is 32B + sizeof(char [])
A Sample String Body with a High Reference Count

(gdb) x/32a 0x7f05c0038fb0
0x7f05c0038fb0: 0xf 0xf
0x7f05c0038fc0: 0x41f 0x726774726f707664
0x7f05c0038fd0: 0x38303d70756f45
0x7f05c0038fe0: 0x7f05c00158

_M_refcount: 0x41f = 1055 instances shared
Windows: 1055 x 40B = ~40KB
Linux: 1*40B + 1054 * 8B = ~8KB

If we had 1M objects: Windows 40MB, Linux 8MB => 32MB delta
⇒ Different platforms utilize memory differently
⇒ Be careful which libraries you use (or roll your own)
Connecting the Dots: A Remote Console story
User wants to view 'console' of a VM

1. User talks to management server
2. Management server locates VM
3. User & VM get connected
The Problem: Remote Console Doesn’t Show Up

- Problem: could not start VM remote console in large environment

- Sequence of debugging
  - Client folks: it’s a server problem
  - Server folks: it’s a client problem
  - Client folks: it’s a ‘vmrc’ problem (vmrc = VMware Remote Console)
  - VMRC folks: authentication? MKS tickets?
  - I got curious…

- More Information: Start remote console for a single VM
  - 50 Hosts, no problem
  - 500 Hosts, no problem
  - 1001 Hosts, PROBLEM!
No Console: Examining the Cases the Actually Work

- **Debugging observations**
  - With < 1000 hosts…
    - Management server CPU and memory goes very high when client invoked
    - Console is dark until CPU and memory go down, then appears
  - Look at server log file
    - Data retrieval call occurs before console appears (WHY???)
    - In failure cases, exception in serializer code
  - Attach debugger
    - Exception is an out-of-memory exception
    - Exception is silently ignored (never returns to client)
No Console: Isolating the Problem

• Problem
  • VMRC creates a request to monitor host information (e.g., is CD-ROM attached)
  • Request gets info on ALL hosts
  • At 1001 hosts, we exceed 200MB buffer on server
  • 200MB restriction only for old-style API clients

• Solution
  • VMRC folks: do NOT create big request
  • Server folks: fail correctly and emit better errors

Nice lessons learned

1. **Create APIs that are difficult to abuse, rather than easy to abuse**
2. **Teach clients how to use APIs**
3. **Make sure (internal) users have input about API design**
4. **Be data-driven in your analysis 😊**
Understanding and using metrics

Memory
Windows-Dev limits/shares example

Windows VM is really slow.

Examples:
- Bootup and login extremely slow.
- Starting up profiling tools (xperf) extremely slow

Starting point in Windows: TaskManager
In-guest metrics

In-guest: memory usage high, but CPU is fine
Going beyond guest-level metrics

We looked in-guest.

What about interaction of this VM with other VMs?

![Diagram of VM interactions]

Windows

OtherOS

VM

VM

Host (running hypervisor)
Memory Primer

VMware ESX hypervisor balances memory of VMs, etc.

- **Page sharing** to reduce memory footprint of Virtual Machines
- **Ballooning** to relieve memory pressure in a graceful way
- **Host swapping** to relieve memory pressure when ballooning insufficient

ESX allows overcommitment of memory

- Sum of configured memory sizes of virtual machines can be greater than physical memory if working sets fit
Ballooning vs. Swapping (1)

Ballooning: Memctl driver grabs pages and gives to ESX

- Guest OS choose pages to give to memctl (avoids “hot” pages if possible): either free pages or pages to swap
  - Unused pages are given directly to memctl
  - Pages to be swapped are first written to swap partition within guest OS and then given to memctl
Ballooning vs. Swapping (2)

Swapping: ESX reclaims pages forcibly

- Guest doesn’t pick pages…ESX may inadvertently pick “hot” pages (→ possible VM performance implications)
- Pages written to VM swap file

Diagram:
- VM1 (guest)
- VSWP (external to guest)
- Swap (w/in guest)
- ESX
- VM2

Steps:
1. Force Swap
2. Reclaim
3. Redistribute
Ballooning vs. Swapping

Bottom line:

- Ballooning may occur even when no memory pressure just to keep memory proportions under control
- *Ballooning is vastly preferably to swapping*
  - Guest can surrender unused/free pages
    - With host swapping, ESX cannot tell which pages are unused or free and may accidentally pick “hot” pages
  - Even if balloon driver has to swap to satisfy the balloon request, guest chooses what to swap
    - Can avoid swapping “hot” pages within guest
Back to my VM: Let’s look at ballooning

VM is ballooning! It reaches its threshold...

Performance Chart Legend

<table>
<thead>
<tr>
<th>Key</th>
<th>Object</th>
<th>Measurement</th>
<th>Rollup</th>
<th>Units</th>
<th>Latest</th>
<th>Maximum</th>
<th>Minimum</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>❌</td>
<td>Windows7-dev</td>
<td>Granted</td>
<td>Average</td>
<td>Kilobytes</td>
<td>6487340</td>
<td>8148800</td>
<td>0</td>
<td>2248651.6</td>
</tr>
<tr>
<td>❍</td>
<td>Windows7-dev</td>
<td>Consumed</td>
<td>Average</td>
<td>Kilobytes</td>
<td>1961788</td>
<td>2363108</td>
<td>0</td>
<td>765293.72</td>
</tr>
<tr>
<td>❍</td>
<td>Windows7-dev</td>
<td>Balloon</td>
<td>Average</td>
<td>Kilobytes</td>
<td>1730272</td>
<td>5452296</td>
<td>0</td>
<td>4209805.9</td>
</tr>
<tr>
<td>❍</td>
<td>Windows7-dev</td>
<td>Swap in</td>
<td>Average</td>
<td>Kilobytes</td>
<td>3103804</td>
<td>3103804</td>
<td>0</td>
<td>1578085.8</td>
</tr>
<tr>
<td>❋</td>
<td>Windows7-dev</td>
<td>Active</td>
<td>Average</td>
<td>Kilobytes</td>
<td>3262584</td>
<td>6543112</td>
<td>0</td>
<td>2743647.0</td>
</tr>
<tr>
<td>❋</td>
<td>Windows7-dev</td>
<td>Swap in rate</td>
<td>Average</td>
<td>KBps</td>
<td>17</td>
<td>3111</td>
<td>0</td>
<td>988.057</td>
</tr>
</tbody>
</table>
Swap-in

And then the VM starts to do host-level swap

Host-level swap impacts performance...
Fine-grained metrics

Check if other VMs are encountering same issue

<table>
<thead>
<tr>
<th>GID</th>
<th>NAME</th>
<th>MEMSZ</th>
<th>GRANT</th>
<th>SZTGT</th>
<th>TCHD</th>
<th>TCHD_W</th>
<th>SWCUR</th>
<th>SWTGT</th>
</tr>
</thead>
<tbody>
<tr>
<td>8283283</td>
<td>VCVA-5.5-bld236</td>
<td>16384.00</td>
<td>14412.00</td>
<td>15374.60</td>
<td>491.52</td>
<td>491.52</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8169268</td>
<td>win2k8-r2-prod2</td>
<td>16384.00</td>
<td>16384.00</td>
<td>16501.00</td>
<td>491.52</td>
<td>491.52</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11021155</td>
<td>vzSim5.1-for-VC</td>
<td>16384.00</td>
<td>16376.00</td>
<td>16061.40</td>
<td>163.84</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>1961387</td>
<td>SLES11</td>
<td>8192.00</td>
<td>6388.00</td>
<td>7097.34</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10967525</td>
<td>Windows7-dev</td>
<td>8192.00</td>
<td>1142.39</td>
<td>1117.20</td>
<td>745.55</td>
<td>430.12</td>
<td>1590.44</td>
<td>1671.02</td>
</tr>
</tbody>
</table>

(esxtop, a top-like utility specifically for ESX hosts)

No other VMs are hitting host-level swapping…

Hmm.

Oh, wait!
Accidentally set limit on VM!

If you set a limit on a VM, it cannot exceed the limit

We configured the VM with 8GB RAM, but set a limit of 1GB!
(btw., this was because I accidentally cloned a VM with a limit…)
(note: our tools track LIMIT, but I didn’t show it on previous slides)
Understanding metrics, part 2

CPU
Hypervisor CPU Scheduling

Run (accumulating used time)
Ready (wants to run, no physical CPU available)
Wait: blocked on I/O or voluntarily descheduled
A customer problem...

Problem

• Customer Performs a Load Test: keeps attaching clients to a server
• At some point, CPU is NOT saturated, but latency starts to degrade
• At some point, client is unusable
• Why?
“Oh yeah, it’s a disk problem…”

Uh-oh! Disk Latencies go over a cliff!

CPU Usage Increases…
Hmm. Not So Fast!!!

Problem:

Yes, Disk Latency gets worse at 4pm. (btw…due to swapping)

However, Application latency gets worse at 3:30pm!

What’s going on from 3:30pm to 4pm?
Looking at a different chart…

<table>
<thead>
<tr>
<th>ID</th>
<th>GID</th>
<th>NAME</th>
<th>NOWLD</th>
<th>%USED</th>
<th>%RUN</th>
<th>%SYS</th>
<th>%WAIT</th>
<th>%RDY</th>
<th>%IDLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>idle</td>
<td>16</td>
<td>111.77</td>
<td>563.57</td>
<td>0.00</td>
<td>0.00</td>
<td>800.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>system</td>
<td>7</td>
<td>0.01</td>
<td>0.02</td>
<td>0.00</td>
<td>700.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>helper</td>
<td>73</td>
<td>0.15</td>
<td>0.25</td>
<td>0.00</td>
<td>7300.00</td>
<td>0.35</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>drivers</td>
<td>9</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>900.00</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>vmotion</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>400.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>console</td>
<td>2</td>
<td>6.45</td>
<td>10.04</td>
<td>0.01</td>
<td>186.59</td>
<td>4.53</td>
<td>86.00</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>vmkapimod</td>
<td>5</td>
<td>0.19</td>
<td>0.28</td>
<td>0.00</td>
<td>500.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>FT</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>vobd.4279</td>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>800.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>net-cdp.4287</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>vmware-vmkauthd</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>100.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>56</td>
<td>56</td>
<td>vm1</td>
<td>5</td>
<td>183.21</td>
<td>248.36</td>
<td>0.28</td>
<td>232.84</td>
<td>18.64</td>
<td>25.13</td>
</tr>
<tr>
<td>69</td>
<td>69</td>
<td>vm2</td>
<td>5</td>
<td>152.17</td>
<td>212.16</td>
<td>0.37</td>
<td>284.93</td>
<td>5.26</td>
<td>77.26</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
<td>vm3</td>
<td>5</td>
<td>126.52</td>
<td>194.13</td>
<td>0.17</td>
<td>302.64</td>
<td>5.64</td>
<td>93.50</td>
</tr>
<tr>
<td>71</td>
<td>71</td>
<td>vm4</td>
<td>5</td>
<td>146.25</td>
<td>219.30</td>
<td>0.21</td>
<td>270.28</td>
<td>11.54</td>
<td>64.30</td>
</tr>
</tbody>
</table>

%Used? %Run? What’s the difference?

%used: normalized to base clock frequency

%run: normalized to clock frequency while VM is running…

%run > %used: Power Management is kicking in…

In this case, turn off power management ➔ latency problems go away
End-to-End Performance

Interactions between hypervisors and guests
Viewing a Video Remotely: Jittery Experience

- interrupt-coalescing-nofix.mov

**Observation: bimodal latencies in 3D graphics workload**
- Needs 80Mbps at peak
- When it reached 80Mbps at peak, dropped down to 30Mbps
- Went back up to 80Mbps
- Dropped to 30Mbps
- Repeat…
Packet transmission in virtualized environment

Guest gives packet to vnic
Guest gives packet to vnic
Packet transmission in virtualized environment

ESX kernel polls queue, sends pkt to pnic
Packet transmission in virtualized environment

After pkt is sent, ESX gives xmit interrupt to guest
When high guest pkt rate is seen, ESX waits for more packets before sending
Coalescing

When high guest pkt rate is seen, ESX waits for more packets before sending
ESX sends transmit interrupt to guest when all packets sent
Our Coalescing Problem:

1. Guest waits for Transmit interrupt before depositing new packet
2. ESX waits for new packet before sending out
After timeout, ESX sends 1 packet
Resulting slow packet rate ➔ ESX disables coalescing
Cycle repeats…
Video playback in Windows: Why Oscillation in Latency?

• Desired Behavior
  • Guest sends packet by giving data to vmnic
  • Hypervisor polls receive queue
  • When packet detected, hypervisor sends packet
  • Hypervisor sends transmit interrupt to guest (packet has been delivered)

• Actual Behavior
  • Hypervisor interrupt coalescing kicks in at high packet rate
  • Guest would not send packet until it received transmit interrupt
  • Both sides wait, timeout in hypervisor, interrupts get sent ➔ drop to 30Mbps
  • Packet rate drops, interrupt coalescing disabled ➔ achieve 80Mbps

• Fix: Windows registry setting to disable waiting for transmit interrupt
Fix for Oscillation in Latency

Fix:

- Known issue in Windows for certain packet sizes
- Disable Windows registry to avoid waiting for transmit interrupt

interrupt-coalescing-withfix.mov

Microsoft KB article:

- [http://support.microsoft.com/kb/235257](http://support.microsoft.com/kb/235257)

VMware KB article:

Bringing it all together
An interesting link

Performance anti-patterns

http://queue.acm.org/detail.cfm?id=1117403

Some examples:

• Fixing Performance at the end of the project
• Algorithmic antipathy: $O(k)$ vs. $O(n)$
• Focusing on what you can see rather than the problem
  • Disk IO is high
    • Option 1 (BAD) Workload needs IO: tell customer to add more spindles
    • Option 2 (BETTER) Find source of IO and eliminate it if possible
• Not optimizing for the common case
Parting Thoughts

Performance debugging is a system-wide exercise

Don’t blindly optimize resources: take a broader view of architecture as well

Don’t take down fences unless you know why they were put up

Make the common case fast (but make sure it is also correct!)