9 Bugs, 4 Lessons, and 1 Turtleneck: Performance Engineering at VMware

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

ravi@vmware.com

MIT Guest Lecture, 6.172

11/27/12
Me, circa 1999

PhD Defense
Me, circa 2012

A small segment in a VMworld Keynote
Email thread from a colleague

“…

Interestingly, as the number of vcbench threads decreased hostsPerThread var increases), the percentage of locktime spent in dbwrites also increases.

…

lots of threads (hostsPerThread = 4):

• ~28 % lock time spent under vdbWrite Connection
• ~16 % lock time spent under exec / commit.

…”

Translation: Why is % lock time in DB increasing despite lighter load?
Step 0: What the ?%##!* is he talking about?

1. Client issues command to server
2. Server performs operation
3. Results persisted to DB
4. Client is notified of completion

Problem: With **lighter** load from client, %time spent in DB Locks **increases**
Examine Lock Hold Time for Various Loads

Latency per lock @ 128 hosts/thread < 4 hosts/thread (Expected…lighter load)
→ Original question: why is %DB increasing with lighter load?
→ Answer: DB latency dominates when overall latency is lower!
**Post mortem on Case Study #1**

1. Understand experimental setup (multi-tier setup)

2. Understand what is being measured (% time in DB lock)

3. Examine relevant data (lock latency)

4. Draw appropriate conclusion
   - Yes, % lock time in DB is higher with a lighter load
   - BUT, overall lock time is small with lighter load
   - Therefore, DB lock time (roughly constant) contributes more to lock latency

*Sometimes, the answer becomes obvious when you state the problem clearly*
Lesson 1: Be Clear

In stating problems, in defining metrics, in demonstrating issues…
Outline

4 Tips for Good Performance Engineering

• Be Clear

• Use the right tool for the right job

• Be Data Driven

• Think Differently (hence, the turtleneck remark…)
Be Clear…In Defining Metrics

- Run (accumulating used time)
- Ready (wants to run, no physical CPU available)
- Wait: blocked on I/O or voluntarily descheduled
Be Clear: What Does This Metric Mean?

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…”

CPU Usage Increases…

Uh-oh! Disk Latencies go over a cliff!
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…

%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

Lesson: be clear in defining your metrics!
Being Clear: Demonstrating a memory problem

Server application runs out of memory after several hours

Use Purify (on a much smaller setup):

- Leak not detected because data was assigned to a reference
- Instead, examine memory in use
  - Do 100 iterations of an operation
  - See 6400B of allocations for an item (100 64B allocations)
  - Code inspection revealed that item was actually not used anymore...a “logical” leak (i.e., there was a free(), but it was never called because the item was thought to be in use)

*If an effect is small, find ways to magnify it. MAKE IT OBVIOUS!!*
Being Clear: narrowing down a networking problem

User issues a request to perform an operation on a VM

• Setup A: Client/Server version 1 to host version 1: 8s
• Setup B: Client/Server version 2 to host version 1: 16s
• Consistent, repeatable difference
• Regression when using new code to talk to older host!

Step 1: Log everywhere

• Client-imposed latency: same in both cases
• Server-imposed latency: same
• Host imposed-latency: extra 8s in Setup B ➔ Focus on the host
Networking Issue: Analyzing the host

Step 2: More logging (standard tools aren’t available on host)

• Narrow down the time...
  ▪ Agent <-> HAL, Setup A: 10ms per call
  ▪ Agent <-> HAL, Setup B: 200ms per call
  ▪ Wow!

Step 3: Examine configuration

• Setup A: named pipe between Agent and HAL
• Setup B: TCP/IP connection between Agent and HAL
Networking Issue: Resolution

Step 4: Solution (intuition by developer)
- Named pipe communication, setup A: 10ms
- TCP/IP communication, setup B: 200ms
- Why? Nagle algorithm on socket connection
  - On a TCP socket, wait for more data before sending packets
  - Can be disabled through TCP_NODELAY option

Step 5: Result
- Use TCP_NODELAY, both have same performance
- Eventually use a cache to avoid interprocess communication

Lesson?
- Once the _real_ problem is clarified, the answer is easier to find
- Client/server code: understand the client/server interaction!
Lesson 2: Use the Right Tool for the Job

...or build your own
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>ntdll.dll!? ::NODOBFM:: `string'</td>
<td>3,109,241.648 ...</td>
<td>76.92</td>
</tr>
<tr>
<td>MSVCR80.DLL!_RTDynamicCast</td>
<td>1,579,519.929 ...</td>
<td>39.07</td>
</tr>
<tr>
<td>MSVCR80.DLL!_RTtypeid</td>
<td>1,529,451.706 ...</td>
<td>37.84</td>
</tr>
<tr>
<td>[Root]</td>
<td>0.01</td>
<td></td>
</tr>
<tr>
<td>MSVCR80.DLL!CxxThrowException</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>[Root]</td>
<td>0.34</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?

Hmm. RTtypeid is used in figuring out C++ type 39% of overall CPU? IncRef and DecRef are main callers.
The Offending Code

```c++
void ObjectImpl::IncRef()
{
    if (_refCount.ReadInc() == 0) {
        const type_info& tinfo = typeid(*this);
        FirstIncRef(tinfo);
    }

    ...
}

Dynamic cast...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)

Lesson: The “Right Tool” can ease debugging dramatically
User complains that server is getting slower and slower

CPU/network/disk not saturated

Memory, however, is increasing dramatically

Eventually, system crashes
Looking at Memory Usage: Perfmon in Windows

Chart of “Private Bytes” for a process vs. time

Memory growing at alarming rate! Not good.

*Private bytes*: memory committed to process (swap space is allocated for it)

Memory given by OS to app, not necessarily memory requested by app (example: fragmentation)

Server is functioning fine, but memory is growing really fast. This could lead to a crash. Let’s investigate…
Profiling Reference-counted Objects

Reference-counted objects

- Pink: mutex
- Teal: condition variable
- Blue: thread activation state

Some thread-related objects increasing

Hmmm... number of threads consumed is also increasing

*** Log files show threads being killed due to uncaught exceptions
Customized Profiling: Pros and Cons

Advantages of our customized profiler:

- Tailored to our application
- Can be made very fast
- Can be run in production environments

Disadvantages:

- Requires code recompilation (then again, so does Purify)
- Specific to this application (code must be refactored for use in other apps)
- Only counts ref-counted objects: what about C code? What about non-ref-counted objects?

*Lesson:* Memory profiling is critical.
Sad Reality: Sometimes, commercial tools don’t work at scale
→ You may have to write your own
Lesson 3: Be Data-Driven in your Decisions

(wherever possible, of course)
Admin wants to clone a VM

1. User talks to management server
2. Management server locates VM on host
3. Host copies VM to another host
More details on customer’s environment

Management server

SAN
Complaint: I upgraded software, and now clone is slower!

Important:
1. Try to change 1 thing at a time
2. Try to do apples-to-apples comparison!
3. Data showed that it wasn’t our software!

1. Customer changed management layer:
   → Clone slowed down
2. However, customer ALSO changed host version
   → New (unqualified) device driver was culprit!
   → Clones to old hosts were actually the same speed
A Colleague wants to get all clusters in a datacenter...

Hierarchy:

DC
  → HostFolder
    → Cluster
      → Host A, Host B, etc.

```python
def GetClustersChildEntity(..., dc):
    hostFolder = dc.GetHostFolder()
    for child in hostFolder.GetChildEntity():
        if isinstance(child, vim.ClusterComputeResource):
            print "\t" + child.GetName()
```
# Use a lower-level API that specifies a container just for clusters

def GetClusters(content):
    # create a container view that contains all clusters in the system
    viewManager = content.viewManager
    filter_targets = [vim.ClusterComputeResource]
    containerView = viewManager.CreateContainerView(content.rootFolder, filter_targets, recursive=True)

    # more to come on next page...

    **WHY DO IT THIS WAY?**
    ME: IT IS FASTER
    HIM: PROVE IT...
A Property Spec?

# skip a bunch of junk and then do the following...

# Create a property spec that says “I care about the cluster name”

# Create property spec

```python
propSpec.SetType(vim.ClusterComputeResource)
propSpec.SetAll(False)
propList = ["name"]
propSpec.SetPathSet(propList)
propSet.append(propSpec)
```

# Skip some junk that specifies

- ‘traversalSpec’ (how to traverse tree) and
- ‘objectSpec’ (what object do I care about [in this case, the containerView])
Finally...AFTER SKIPPING A BUNCH OF STEPS

```python
pfs.SetObjectSet(objSet)  # what objects?
pfs.SetPropSet(propSet)  # which properties?
pfса = [pfs]

... retrieveResult = pc.RetrievePropertiesEx(pfса, retrieveOptions)
    for object in retrieveResult.objects:
        dps = object.GetPropSet()
        VM = object.GetObj()
        if (dps != None):
            for dp in dps:
                if (dp.name == "name"):
                    print "\t" + dp.val
```
Whoa…what did we just do?

The ‘faster’ way:

• 30% faster on small inventories
• More than 3x faster on bigger inventories
• (reduced RPC calls, better CPU/network utilization, etc.)

• But…

• REALLY, REALLY PAINFUL…
• NO ONE WOULD DO IT THIS WAY WITHOUT PROOF IT IS BETTER!

• Lessons:
  • Be data driven
  • Please, please, design scalable and easy-to-use APIs 😊
  • (i.e., don’t expose underlying implementation details)
Lesson 4: Think Differently

(within reason...)

45
Think Differently…the setup

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?
  - Me: this is ridiculous…THINK DIFFERENTLY

- 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

Lesson 0: Think differently. Other 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 😊
Lesson 4a: _Really_ Think Differently 😊

(within reason...)
A Social-Media Approach to Virtualization Management

Ravi Soundararajan (ravi@vmware.com)

Performance Group
This presentation may contain product features that are currently under development.

This overview of new technology represents no commitment from VMware to deliver these features in any generally available product.

Features are subject to change, and must not be included in contracts, purchase orders, or sales agreements of any kind.

Technical feasibility and market demand will affect final delivery.

Pricing and packaging for any new technologies or features discussed or presented have not been determined.

This information is confidential
The Million Dollar Question

My Job: Hundreds of VMs

My Life: Hundreds of Friends

Which is easier to manage?
An example of management...a Datastore has gone offline...

Shared NFS Datastore

ESX Hosts
An Example: How one tool shows the issue

Hosts that are affected
(How widespread is the problem?)

Error messages
(which server?)
The Socialcast view of the same issue

Error: Lost NFS connection to 10.135.193.155
Also, 13 other servers are affected
Useful Attributes of Social Networking

Reduction of information
• 13 likes vs. 13 similar messages

Hierarchy and connectivity
• Follow a subset of people, not the world
  (more on this later)

Domain relevant alerting
• Can set alerts on stuff important to humans, like birthdays
  (more on this later)
Applying These Principles to Virtual Infrastructure

VMs/Hosts post relevant status messages

Like/flag messages instead of re-posting

Message streams and groups for aggregation
Reduction of Information and “Relevant” Messages

3 Likes vs. 3 distinct messages for same problem (in-guest disk usage > 80%) ➔ reducing information

Disk Usage high: relevant & critical for certain VMs (e.g., VC DB 😊)
Which VM users are affected if vCenter goes down?

VI Admin (me) linked to end users (who are managed by IT) from one portal
Prototype Details

1. Give hosts/VMs/vCenters/Admins a Socialcast account
2. Socialcast REST-based API (likes, follows, search, groups, messages)
3. vCenter and VMs use Curl to communicate with Socialcast
4. Service VMs acts on behalf of ESXi host (no Curl on ESXi)
Conclusions

Use techniques from social media to ease virtualization management

• Information reduction

• Hierarchy and connectivity easily derivable

• Ability to provide relevant messages and alerting
Future Work

Sending commands to VMs via Socialcast (already done)

Generalized query framework via Socialcast ("Select VMs with XYZ logged in")

“Smart Groups”: another means of data summarization
- Misbehaving VM (e.g., in-guest swap): adds itself to ‘thrashing VMs’ group, removes itself when condition disappears
- Any VI Admin can see which VMs need help by looking at ‘thrashing VMs’ group
- Audit trail of changes is in Socialcast

Management from mobile device

Coupling to other data sources (e.g., Analytics like VCOps)
Potentially Interesting Research Questions

• Performance of Graph Databases

• Mapping “Social constructs” to Virtualization Management (e.g., switches, disks)

• Performance of Interpreted Languages (Ruby on Rails, Python)

• Data “gnostic” Health Monitoring of Virtualized Datacenters
Conclusion to the actual talk 😊

- Be Clear: communication helps you and it helps others

- Use the right tool for the right job, or build your own 😊

- Be Data Driven: “I think…” is not usually good enough

- Think Differently (hence, the turtleneck remark…)
  - Be willing to be different…but
  - Be willing to admit if you are wrong