ORACLE VM

An Oracle White Paper
December 2015

Oracle VM 3:
10GbE Network Performance Tuning

ORACLE

Introduction

This technical white paper explains how to tune Oracle VM Server for x86 to effectively use 10GbE (10 Gigabit Ethernet) networks. This document applies to Oracle VM 3 and is mostly but not exclusively focused on multi-socket servers. These servers are popular in today's deployments and have NUMA (Non-Uniform Memory Access) latency effects that affect network performance.

Network performance considerations

Network performance is an essential component of overall system performance, especially in a virtual machine environment. Networks are used in separate roles: for live migration, cluster heartbeats, access to virtual disks on NFS or iSCSI, system management, and for the guest virtual machines own network traffic. These roles, and other aspects of Oracle VM networking, are explained in Looking "Under the Hood" at Networking in Oracle VM Server for x86.

10GbE network interconnects are increasingly deployed in modern datacenters to enhance performance for network-intensive applications. It can be challenging to achieve 10G line rate or otherwise achieve expected throughput. This is especially the case when using a single stream of transmit/receive. The difficulty is increased with virtualized systems, which typically add overhead and latency for virtual I/O.

This is particularly challenging on systems with many CPUs and large RAM capacity. These systems are very attractive for consolidating multiple smaller physical servers into virtual machines, but they have NUMA (Non-Uniform Memory Access) properties in which the memory latency (time to access memory locations in RAM) can vary widely depending on the CPUs and memory locations being used. This can reduce performance and scalability for high-speed networks whose inter-packet times make them sensitive to variation in memory latency. Performance can vary depending on whether interacting processes and kernel components, and the RAM locations they access, are closely aligned on the same CPU node (socket) or core. Affinity to the same CPU nodes reduces memory latency. It is important to know the CPU topology and capabilities of the system to optimally tune for particular application performance needs.

An Example Illustrating NUMA effects

The following example illustrates 10G network performance on two different hardware architectures.

Netperf TCP_STREAM was used for measurement. The initial result was before applying any tuning to improve network performance. A Linux virtual machine with 2 VCPUs and 2GB of memory was used for all tests. Network performance was measured between dom0 instances on two machines (with direct access to the physical network adapters), and between a dom0 on one server and a guest virtual machine on the other server.

- Non-NUMA systems with adequate system resources.

  - Two systems, each with 2 sockets, 4 CPU cores per socket, and 2 threads per core (a total of 16 CPU threads), and 24GB of memory
  - dom0 with 8 VCPUs, and 2G of memory
  - dom0 -> dom0 : ~9.4Gb/second
  - dom0(a) -> a VM running on dom0(b) : ~7.8Gb/second
- NUMA systems with plenty of resources
  - Two systems, each with 8 sockets, ten CPU cores per socket and 2 threads per core (160 CPUs total), and 256GB of memory
  - dom0 with 160 VCPUs, and 5GB memory
  - dom0 -> dom0 : ~5.2Gb/second
  - dom0(a) -> a VM running on dom0(b) : ~3.4 Gb/second

Results on untuned large systems are low compared to the non-NUMA systems running the same software on servers with fewer CPUs and less memory. Throughput between dom0 environments was reduced by 45%, and between dom0 and a guest on the adjacent server reduced by 56%. This illustrates poor scalability when tuning is not provided on machines that are nominally the same speed.

Performance Tuning

Several important tuning changes were made to improve performance on the larger machines. These changes reduced memory latency by restricting CPU scheduling and interrupt processing to a subset of the available CPUs. Other changes adjusted TCP parameters with values for fast networks: those changes are appropriate for both small and large servers.

Note: use these changes as guidelines and adjust for your hardware architecture and resources. They have not been exhaustively tested on a wide range of configurations and workloads. Changes should be individually evaluated in the context of performance requirements, with consideration of their effect on maintainability and management effort. These changes include:

Number of dom0 CPUs

On a large 8-socket x86 system with NUMA properties, the number of dom0 VCPUs was reduced from 160 to 20, which is the number of CPU threads on the first socket of the system. This limits CPU scheduling on dom0 to the first CPU socket, which ensures local memory access and L2 cache affinity. dom0 VCPUs were pinned so virtual CPUs had strict correspondence to physical CPUs (VCPU0 -> CPU0 .... and so on).

The number of CPUs on a socket depends on the server's CPU topology, which can be determined by logging into the server and issuing the command "`xm info`". This displays the number of `cores_per_socket` and `threads_per_core`. The product of these values is the number of CPU threads per socket to be used in the step below.

To pin and change dom0's VCPUs add:

"dom0_vcpus_pin dom0_max_vcpus=X" to the Xen kernel command line (in this case, X=20), and reboot. In this example, the complete line from /boot/grub/grub.conf would be

```
kernel /xen.gz dom0_mem=582M dom0_vcpus_pin dom0_max_vcpus=20
```

This had the largest effect of any of the tuning procedures described in this document. Starting with Oracle VM Server 3.2.2, a newly installed dom0 is configured with a maximum of 20 VCPUs to provide better "out of the box" performance for large systems. There is nothing special about that number, though it covers most servers. The precise recommendation is to set the number of dom0 virtual CPUs to at most the number of physical CPUs per socket, and pin dom0 virtual CPUs to physical CPUs on the first socket. If hyper-threading is being used, count in units of threads rather than cores.

Port Interrupt Affinity

Most 10G ports use MSI-X interrupt vectors. These are spread over available dom0 VCPUs, and at least one of them is associated with CPU0. Since CPU0 is the default interrupt CPU for many devices, interrupts can arrive on the 10G port while the CPU is servicing an interrupt from another device and cannot be preempted. This delays handling the network interrupt until the other interrupt service routine finishes. It may be useful to move 10G port interrupt vectors to CPUs that are less busy servicing interrupts. This example shows how to change interrupt affinity.

1. Find the IRQ IDs for the ports by issuing "cat /proc/interrupts| grep ethX", replacing ethX with the name of the network ports. The first column has the IRQ ID. An example of what this looks like (for a smaller server that fits on the page) is shown below:

   ```
   # cat /proc/interrupts|head -1; cat /proc/interrupts|grep eth
              CPU0       CPU1       CPU2       CPU3
    35:    3562070        243    1307766        249  PCI-MSI-edge eth4
    39:   11828156     108367     101470     108262  PCI-MSI-edge eth0
    40:   11748516   11991143   11688127   11988451  PCI-MSI-edge eth1
   ```

2. Find which CPU each interrupt vector is associated with. If you have only one such port, just "cat /proc/irq/$irqid/smp_affinity", replacing $irqid with the value from step 1. If you have many ports you can use a script like the following.

   ```
   #!/bin/bash
   for irqid in `cat /proc/interrupts| grep eth | awk
   {'sub(":","",$1);print $1'}`;
   do
    affinity=`cat /proc/irq/$irqid/smp_affinity`
    echo "irqid $irqid affinity $affinity"
   done
   ```

This types out the IRQ number and the bit mask that represents which CPUs are associated with that interrupt, with the rightmost bit representing CPU0. For example, a value of 1 would indicate CPU

0, and a value of "f" would indicate CPUs 0 to 3. If you have an IRQ assigned to CPU 0, you can reassign it to a less-busy CPU by replacing the contents of smp_affinity with a new bit mask. For example, the following command sets the CPU affinity for IRQ 323 to CPU 2:

```
# echo 00004 > /proc/irq/323/smp_affinity
```

This mechanism can be used to statically load-balance interrupt processing, and can be useful if a particular CPU is unevenly loaded. Alternatively, you can use the `irqbalance` service which has similar function but does not assign specific IRQs to specific CPUs, which can be helpful to force interrupts to the same CPU core. If using static allocation as described above, turn off the `irqbalance` service by issuing "`service irqbalance stop`"

This change should be carefully evaluated since it involves changing Linux kernel settings. Administrators should consider whether the change is necessary to achieve performance objectives, and whether CPUs are heavily loaded due to interrupt processing. Installation standards and comfort level regarding modifying kernel settings must also be observed.

Process Affinity

In NUMA systems it is advisable to run interacting processes (in this case Xen's Netback and Netfront) on the same CPU socket. This helps reduce memory latency for accessing shared data structures. To implement this, pin a VM's virtual CPUs to physical CPUs on the same socket as where dom0 runs. This should be done selectively for the most performance-critical VMs to avoid putting excessive load on the first server socket's CPUs.

The recommended method is to use the Oracle VM 3 Utilities `ovm_vmcontrol` command to obtain and change the virtual CPU settings. The Oracle VM 3 Utilities are provided as an optional add-on to Oracle VM 3. Documentation and download instructions can be found at the Oracle VM Downloads page at http://www.oracle.com/technetwork/server-storage/vm/downloads/index.html

After installing the utilities, log into the server hosting Oracle VM Manager, and then issue commands as follows, replacing the variables for administrator userid, password, and guest VM name as needed:

```
# cd /u01/app/oracle/ovm-manager-3/ovm_utils

# ./ovm_vmcontrol -u $ADMIN -p $PASSWORD -h localhost \

            -v $GUEST -c vcpuget
```

That displays the set of physical CPUs the guest can run on as "Current pinning of virtual CPUs to physical threads". If no CPU pinning has been done, this will be the total list of CPUs on the server. That is generally larger than the number of virtual CPUs in the guest (for example, a server with 8 CPUs hosting guests with as few as one VCPU). For example:

```
# ./ovm_vmcontrol -u admin -p Welcome1 -h localhost -v myvm -c vcpuget

Oracle VM VM Control utility 2.0.1.

Connected.

Command : vcpuget

Current pinning of virtual CPUs to physical threads : 0,1,2,3,4,5,6,7
```

To pin the guest domain to the physical CPUs on the first CPU socket, issue the following command, replacing N with 1 less than the number of CPUs per socket (because CPUs are counted starting from zero):

```
# ./ovm_vmcontrol -u $ADMIN -p $PASSWORD -h localhost \
            -v $GUEST -c vcpuset 0-N
```

On a server with 20 CPUs per socket the value would be `vcpuset 0-19`. An example is:

```
# ./ovm_vmcontrol -u admin -p Welcome1 -h localhost \
            -v myvm -c vcpuset -s 0-19
Oracle VM VM Control utility 2.0.1.
Connected.
Command : vcpuset
Pinning virtual CPUs
Pinning of virtual CPUs to physical threads '0-19' 'myvm' completed
```

In this example we pinned the guest to a number of physical CPUs larger than the number of virtual CPUs. That lets the OVM scheduler dispatch the guest on any of the physical CPUs on the first socket. You can optionally specify physical CPUs to exactly match the number of virtual CPUs. This might improve Level 1 cache hit ratios, but could also lock the guest onto heavily loaded physical CPUs when there are less-utilized CPUs available.

Another way to specify physical CPUs, used if the Oracle VM 3 Utilities are not installed, is to locate and edit the VM configuration file by hand. First, use the Oracle VM Manager user interface to display the ID for the Repository and virtual machine. Then login as root to one of the servers in the pool and edit the file vm.cfg in the directory /OVS/Repositories/$RepositoryID/VirtualMachines/$VMID Change the "cpus" line in vm.cfg as follows:

```
vcpus = 2
cpus = "1,4"
```

This will pin the VM's 2 VCPUs to CPU1 and CPU4. Note that the command "xm vcpu-list" can be used to verify that domains have the correct number of CPUs and pinning, including for dom0. The `ovm_control` command is recommended because it is a documented method rather than "out of band" editing of internal file contents.

In these examples we pin guest virtual CPUs to the first socket along with dom0. That prevents NUMA overhead, but also prevents using other sockets. If this is done for all guests it would eliminate the benefit of using a server that scales to multiple sockets. Therefore, this technique should be used selectively for virtual machines with the most critical network performance requirements. Other guests can run on other CPU sockets without CPU pinning and experience better performance than if all are assigned to an overloaded first socket. Pinning CPUs is also an administrative and management overhead that should not be done indiscriminately.

NUMA latency and cache misses within a virtual machine can reduce performance, independent of network concerns, so it is important to size a virtual machine to not have an excessive number of CPUs. One best practice is to size for expected CPU loads plus some headroom. Oracle VM lets you set the number of CPUs the guest receives when it starts, and a maximum number you can adjust it to during exceptional loads. That makes it less likely that virtual CPUs straddle CPU sockets and experience NUMA latency, increases reuse of warm cache contents on CPU cores, and reduces internal lock contention for data structures that have to be serialized over multiple CPUs.

Power States

Many servers offer the ability to run with lower power consumption and generate less heat when idle or at low loads. Some servers take a long time to recover from power savings mode, which can affect latency service network activity, so it may be worth disabling sleep modes ("C-states") or avoid using low processor power consumption mode ("P-states"). This is a server-dependent option that should be evaluated for each specific server model.

TCP Parameter Settings

Default TCP parameters in most Linux VM distributions are conservative, and are tuned to handle 100Mb/s or 1Gb/s port speeds, and result in buffer sizes that are too small for 10Gb networks. Modifying these values can lead to significant performance gain in a 10Gbs VM network. RTT, and BDP are essential values influenced by TCP parameters. Round Trip Time (RTT) is the total amount of time that a packet takes to reach a destination and get back to the source. In a Gigabit network this value is between 1 and 100ms. RTT can be measured using ping. Bandwidth Delay Product (BDP) is the amount of data that can be in transit at any given time. It is the product of the link bandwidth and the RTT value. Assuming 100ms RTT:

BDP = (.1s * (10 * 10^9)bits) = 134217728 Bytes.

Buffer sizes should be adjusted to permit the maximum number of bytes in transit and prevent traffic throttling. The following values can be set on Linux virtual machines as well as dom0.

```
Maximum receive socket buffer size (size of BDP)
# sysctl -w net.core.rmem_max=134217728

Maximum send socket buffer size (size of BDP)
# sysctl -w net.core.wmem_max=134217728

Minimum,initial,and max TCP Receive buffer size in Bytes
# sysctl -w net.ipv4.tcp_rmem="4096 87380 134217728"

Minimum, initial, and max buffer space allocated
# sysctl -w net.ipv4.tcp_wmem="4096 65536 134217728"

Maximum number of packets queued on the input side
# sysctl -w net.core.netdev_max_backlog=300000

Auto Tuning
# sysctl -w net.ipv4.tcp_moderate_rcvbuf=1
```

Figure 1: Use the above commands to adjust TCP parameters

These settings take effect immediately but do not persist over a reboot. To make these values permanent, add the following to `/etc/sysctl.conf`

```
net.core.rmem_max = 134217728
net.core.wmem_max = 134217728
net.ipv4.tcp_rmem = 4096 87380 134217728
net.ipv4.tcp_wmem = 4096 65536 134217728
net.core.netdev_max_backlog = 300000
net.ipv4.tcp_moderate_rcvbuf =1
```

Figure 2: Add these lines to `/etc/sysctl.conf`

Additionally, for Oracle VM Server for x86 prior to version 3.2, netfilter should be turned off on bridge devices. This is done automatically in version 3.2 and later.

```
# sysctl -w net.bridge.bridge-nf-call-iptables=0
# sysctl -w net.bridge.bridge-nf-call-arptables=0
# sysctl -w net.bridge.bridge-nf-call-ip6tables=0
```

To see what settings are in effect, type `sysctl` followed by the parameter name, for example:

```
# sysctl net.core.rmem_max
```

Jumbo Frames

The default MTU value is 1500, and most 10G ports support up to 64KB MTU values. An MTU value of 9000 was adequate to improve performance and make it more consistent. Jumbo frames have been supported in Oracle VM Server for x86 starting with Release 3.1.1. Remember that if you change the MTU, the changed value must set be set on all devices (like routers) between the communicating hosts. Note that some switch configurations preclude use of jumbo frames (such as QoS, trunking or even VLANs), depending on switch vendor or model.

NIC Offload Features

Offload features supported by the NIC can reduce CPU consumption and lead to a significant performance improvements. The settings below show values used during testing. Note that Large Receive Offload (LRO) must be left in its default state. If turned on, it will be automatically set to `off` when a port is added as a bridge interface. An example of these settings is shown below.

```
# ethtool -k ethX
Offload parameters for ethX:
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp-segmentation-offload: on
udp-fragmentation-offload: off
generic-segmentation-offload: on
generic-receive-offload: on
large-receive-offload: off
```

Figure 3: Illustration of ethtool settings

Results

The following results were observed after applying the above tuning options.

dom0 -> dom0: ~9.5Gb/s – This is very close to wire speed and the baseline non-NUMA value, and 83% faster than the baseline NUMA value.

dom0(a) -> a VM running on dom0(b): ~8.2Gb/s – 5% faster than the non-NUMA baseline, and 240% faster than the baseline NUMA value.

VM on one host -> a VM on a different host: ~8.2Gb/s – showing high bandwidth even going through two bridged connections.

Depending on your system architecture, network, resources, and application requirements, you may find that not every recommendation in this paper is needed. On NUMA systems, the most important action is reducing the number of dom0 VCPUs so they all fit on one CPU socket/node.

Summary

Network performance is a critical component of total system performance, and can vary greatly depending on system configuration and tuning options. Performance on large systems with NUMA behavior can be dramatically improved by applying simple tuning procedures. These tuning methods can be used with Oracle VM Server for x86 3.1.1 and later.

There is no single set of tuning rules that works in all situations. Tuning recommendations must be balanced against other aspects of total system performance, and tested in realistic configurations to determine actual performance effectiveness. It may be sufficient to apply a few of the tuning procedures listed and then stop when acceptable performance is achieved. Tuning has to be carefully evaluated in the context of installation standards, supportability, and administrative best practices. With these controls in place it is possible to substantially improve performance.

Oracle VM 10GbE Network Performance Tuning
December 2015
Authors: Adnan Misherfi, Jeff Savit

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Integrated Cloud Applications & Platform Services