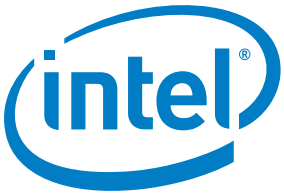


Real-Time Virtualization on an Intel® Xeon® Server

White Paper

July 2015

Document Number: 332126-002

Contents

Figures

Tables

Revision History

Date	Revision	Description
July 2015	002	Updated Section 8.0.
March 2015	001	Initial release.

§

1.0 Introduction

Running deterministic work load on a virtual machine requires much more than the right hardware and software components. The configuration, integration, and interoperability of the components determines whether the system can run multiple real-time tasks simultaneously.

This white paper shares insights into the impact of various components on determinism in task execution. This white paper also explains the pros and cons of making those components suitable for real-time. The components are:

- BIOS
- System management interrupts
- Linux* real-time kernel patch
- Interrupt Request (IRQ) affinity
- Isolating the CPU from the Linux scheduler
- Isolating the Read Copy Update (RCU) (Linux synchronization mechanism) from the Linux scheduler
- Configuring the QEMU

Further, this white paper also provides setup details and the results of applying these changes on a host and a virtual machine on the Intel® Xeon® E5-2699 v3 server in terms of IRQ latency.

1.1 Terminology

Table 1. Terminology

Term	Description
ASPM	Active State Power Management
CAT	Cache Allocation Technology
CMT	Cache Monitoring Technology
IOMMU	Input/Output Memory Management Unit
IRQ	Interrupt Request
LLC	Last Level Cache
NFS	Network File System
OVP	Open Virtualization Profile

Term	Description
QEMU	Quick Emulator
RCU	Read Copy Update
SELinux	Security Enhances Linux
SMI	System Management Interrupts

§

2.0 Technical Challenge

Intel® x86 processors are designed for effective execution of general purpose applications. This means, the loads such as processing, memory, and I/O will be evenly distributed as much as possible and will be successful almost every scenario. This is an excellent design for general purpose application, but will not suit a real-time application.

A real-time application is all about doing “x” amount of work in duration “y”. There are mainly two types of real time: soft and hard real times. In hard real time, it is mandatory to finish the task in the specified time, every single time. In soft real time, data loss is tolerated to a certain level. The optimizations for general purpose applications can cause variation in the duration in doing a fixed amount of processing. There will also be few outliers, where the time is extremely high. This is not acceptable in real-time tasks. When virtualization is included, this becomes much more complex scenario. The duration of outliers can exponentially increase in a virtualized system.

§

3.0 Solution

An Intel system designed with the right software components and configured to optimize the workload can eliminate the spike in processing time as well as optimize the performance of real-time tasks, even if the task is running in a virtual machine. This optimization starts in BIOS tuning and goes through activities such as Linux kernel RT patching and optimization, kernel command line optimization for isolating CPU cores, Linux RCU, and interrupt affinity. The system can be purposed for performance with minor compromises in features by tuning various parameters explained in following sections.

§

4.0 BIOS

The BIOS should be the first component to be optimized. Here are the options that can be tweaked as per the requirements in the system design.

4.1 Processor Optimizations

1. Intel® Turbo Boost Technology - Turbo mode increases the frequency of certain cores and in-turn affects the performance of other cores. This can cause a higher variation in time taken to do a fixed amount of processing. Turning turbo mode off will help to reduce the variation in frequency, and in turn latency.
2. Intel® Hyper-Threading Technology (Intel® HT Technology) - Turning hyper-threading off will dedicate entire resources for the core, thus giving better determinism than executing a task with this option turned on.
3. CPU C-States – CPU C-States are sleep states. This works primarily by turning on/off certain components of the CPU. A lower C-State gives a lower sleep ratio, thus improving latency. Turning C-States off or limiting them will improve determinism.
4. CPU P-States – CPU P-States are power states. These work on reducing voltage and frequency of processor cores. Limiting P-States will limit the change in frequency, thus improving determinism.
   If power management is mandatory in the system design, the first option to consider is using a fixed P-state.
5. Intel® Virtualization Technology (Intel® VT) for IA-32, Intel® 64 and Intel® Architecture (Intel® VT-x) represents the hardware virtualization features of the processor. Ensure to enable it if virtual machines are used in system design.

4.2 Memory Optimizations

1. Memory power management can fluctuate the frequency. Keeping constant the frequency will improve performance. Hence, disabling memory power management should improve performance.
2. If permitted by the BIOS settings, set the memory frequency to the maximum supported by both the DIMM module and the mother-board. This further enables to set a constant high frequency.

4.3 I/O Optimizations

1. If the bottle-neck is in I/O bandwidth, try disabling PCIe* Active State Power Management (ASPM) and verify the performance. This disables the PCIe link power management.
2. The feature, Intel® Virtualization Technology (Intel® VT) for Directed I/O (Intel® VT-d), enables enhancement for I/O related virtualization. Ensure to turn Intel® VT-d on, if virtual machines are used in the system.
3. Turn off the USB if this feature is not used. If not, selectively disable unused features such as boot from USB.

4.4 Miscellaneous

There are certain System Management Interrupts (SMI) that cause a spike in the time taken to do “x” amount of processing. Some BIOS provided give a provision to selectively disable certain SMIs. If the BIOS has this provision, disable the SMI related to the features that are not being used.

§

5.0 Linux Kernel

The first step to enable real-time performance in the operating system is to ensure that the Linux kernel is patched with the appropriate preempt-RT patch from https://www.kernel.org/pub/linux/kernel/projects/rt and get the equivalent kernel from https://www.kernel.org/pub/linux/kernel/. Refer to https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO for patching instructions. Further fine tuning can be done by disabling the components that are not being used.

5.1 Linux Kernel Additional Patches

All user patches are not up-streamed to the Linux kernel. Many of them are rejected. In some situation, a patch rejected by the mainstream may suit the technical requirement. In other cases, a patch may exist for the change that you are planning to implement. Hence, it is always a good idea to browse through https://lkml.org/ and other sites for the patches. The patch may have to be ported to the version of Linux used in the system.

§

6.0 Dedicated CPU Cores

Dedicating a CPU core for a real-time task is one of the most important aspect for consistent performance. There is a general misconception that isolating CPU cores from the Linux scheduler will take care of this problem. The scheduler isolation is just the first step. Here are the steps, beginning with CPU core isolation.

6.1 CPU Core Isolation

This method requests the Linux scheduler not to use certain CPU cores general kernel SMP balancing and scheduler algorithms. The Linux scheduler on an RT patched kernel accepts the request almost every time. The exception, if any, is to execute a critical kernel code block that affects system stability.

The isolation is done by adding `isolcpus` to GRUB (Linux boot loader). For example, in an 8 core system (core 0-7), if cores 1-7 have to be isolated, the parameter should be `isolcpus=1-7`.

6.2 Linux RCU Isolation

Isolating tasks from certain CPU cores does not isolate all activities on that core. Another activity that runs on all CPU cores is Linux Read Copy Update (RCU). RCU is a synchronization mechanism in the Linux kernel. Refer to https://www.kernel.org/doc/Documentation/RCU/whatisRCU.txt to know more about RCU.

RCU can be isolated by adding the two parameters `rcu_nocbs` and `rcu_nocb_poll` to the GRUB kernel initialization command. When using the same example of the previous section, add `rcu_nocbs=1-7, rcu_nocb_poll` to the GRUB config.

6.3 IRQ Affinity

The Interrupt Request (IRQ) is another activity that runs on all CPU cores. The IRQ affinity can be changed at runtime. To change the IRQ 10 affinity to 0, issue a command `echo 0 > /proc/irq/10/smp_affinity`. An IRQ used by a real-time task can be assigned to a specific isolated core(s) using the same method. Further, modifying `/proc/irq/default_smp_affinity` along with disabling/tweaking `irqbalance` will set the default IRQ mask for new threads created from then onwards. These steps can be automated by adding a custom script in `/etc/rc.local`. The link https://www.kernel.org/doc/Documentation/IRQ-affinity.txt provides more information on IRQ affinity.

§

7.0 Linux Power Management

The power management can be altered by Intel specific drivers in Linux. Hence, setting power management options in the BIOS may not give the desired result. The sub-topics in this section explain how to control power management in Linux.

7.1 CPU C-States Enforcement

The CPU C-States can be changed from its BIOS settings by the power management driver `intel_idle` present in the Linux kernel. Unlike other OS and legacy power management drivers, this driver does not use the ACPI tables to determine the C-State. The current state can be verified using the utility `powertop`. The CPU C-State can be explicitly set in the GRUB by the `intel_idle.max_cstate` kernel parameter. For example, setting `intel_idle.max_cstate=0` in the GRUB will disable the C-State transitions per core globally. Further, `processor.max_cstate=1` and `idle=poll` will reduce latency caused by the C-States. For the list of other Kernel parameters, refer to https://www.kernel.org/doc/Documentation/kernel-parameters.txt.

7.2 CPU Frequency Consistency

Keeping a constant CPU core frequency will result in consistent performance. This can be achieved by setting the configuration `scaling_min_freq` and `scaling_max_freq` in `/sys/devices/system/cpu/cpu`**`X`**`/cpufreq` to the same value. This in conjunction with the Linux power governors will give the desired result. The power governor `performance` is recommended for constant high performance and the power governor `userspace` is recommended for run-time changes.

If there is a requirement for P-State tuning, follow the instructions at https://www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt.

§

8.0 Platform Quality of Service

Cache Monitoring Technology (CMT), Cache Allocation Technology (CAT) and Memory Bandwidth Technology (MBM) are a family of technologies designed to monitor and partition shared resources such as the last-level cache. For a full list of SKUs that support these technologies, see: https://www-ssl.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html.

Cache and memory sensitive real time applications may benefit from CAT, specifically those workloads that have a smaller footprint that fit in the last-level cache, can be protected utilizing the cache allocation technology. For instance, an interrupt service request that is used by a real-time task can be kept in a last-level cache partition.

For additional information regarding CAT, refer to Section 17.15 in Volume 3 System Programming Guide of the *Intel® 64 and IA-32 Architectures Software Developer's Manual* available at: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html.

§

9.0 Other Kernel Parameters

There are few other kernel parameters that improve the performance of the host machine, especially in the host. This is done by adding them in grub configuration

9.1 Intel Input/Output Memory Management Unit (IOMMU)

In order to improve the performance of the Intel IOMMU, ensure that `intel_iommu=on` is added to the GRUB config. This ensures that the Intel IOMMU feature is forced on. If the device pass-through is used, add `iommu=pt` to the GRUB config to get the best performance.

9.2 SELinux

SELinux stands for Security Enhances Linux. SELinux is a Linux kernel security module that provides a mechanism for supporting access control security policies. This consumes processing time. If the system design does not require SELinux, disable it. This can be done by editing the SELinux config. The location of the SELinux config can be different in different distributions.

§

10.0 VM Specific Changes

A few VM specific changes will help to improve the performance. These tweaks can be implemented if the system design allows it.

10.1 NFS-Based File System on Host

Creating an NFS-based file system for the VM on the host itself will improve the performance of file read/writes. This will remove the need of updating the file allocations twice if there is an expansion of the file system or if the entire file system image has not been created.

10.2 Locking Memory Allocated for VM

There are certain patches for Quick Emulator (QEMU) that locks the memory on a VM launch. This will let the memory allocated for the VM to stay in the DRAM without being swapped out. The link, http://lists.gnu.org/archive/html/qemu-devel/2012-09/msg05081.html, provides this patch. There are similar patches that exist in the recent versions. If the QEMU version used in the system design does not have this feature, it is advisable to port and apply this functionality.

10.3 VM Virtual CPU to Host Physical Pinning

A one-to-one PCPU-to-VCPU mapping can improve performance. One of a few different methods to achieve this is by using the command `taskset` and assigning a VM thread to a core after the VM is launched. The QEMU threads PID can be obtained by the command ps -eLf | grep qemu on the host. Then that PID can be attached to a physical core by taskset -p <mask> <PID>.

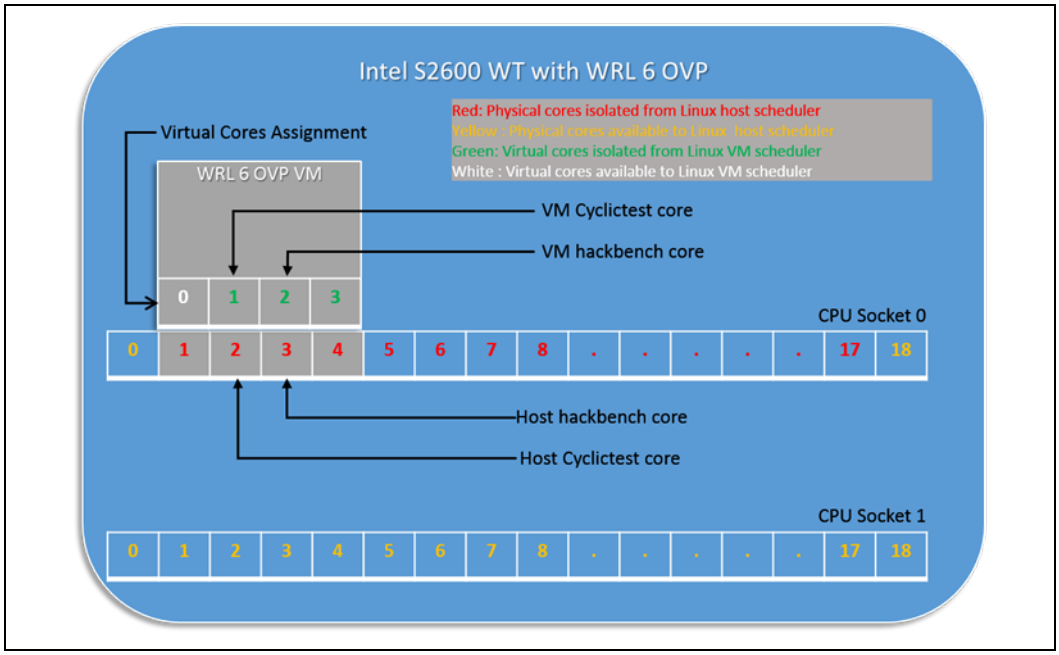
§

11.0 Test Setup

A test setup was used to try many of these options. The platform used was the Intel® Server Board S2600WTT with two Intel® Xeon® CPU E5-2699 v3 at 2.30 GHz and 64 GB DDR4. This latest Intel® Xeon® processor has 18 physical cores (without hyper-threading). This means that there is sufficient physical cores for all the real-time applications. Further, one core was dedicated for the host and guest operating system.

The host and guest operating system was Wind River* Linux 6 OVP (preempt RT based), tuned with the settings mentioned in this white paper. The objective of this test setup was to demonstrate that a real-time task is not affected by load running on another core. This was done by measuring the IRQ latency on the host and VM using the publically available benchmarking tools. In this setup, the utility `cyclictest` was used for measuring the IRQ latency while the utility hackbench was used as the load. These utilities were part of the RT-tests package available at https://www.kernel.org/pub/linux/kernel/people/clrkwllms/. The utility `cyclictest` generated an interrupt periodically and the minimum, maximum, and average IRQ response time is measured. All tests were run for 24 hours.

Figure 1. Intel® Server Board S2600WTT



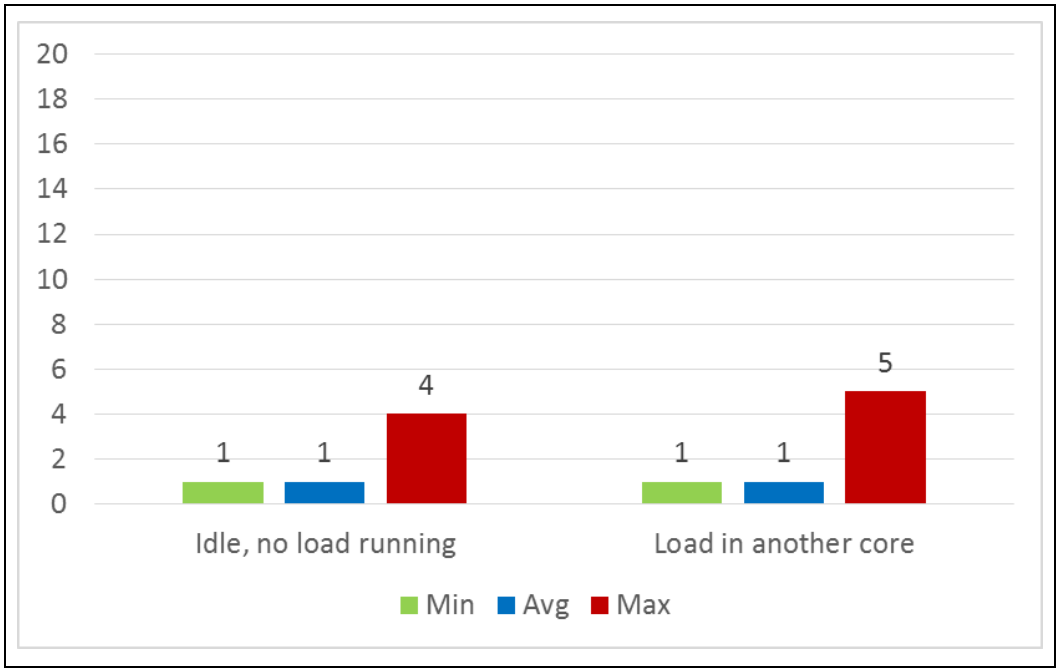
§

12.0 Results

12.1 Result on Host

The OS has a dedicated core for all generic activities. The load hackbench ran on one core on the host (isolated from the Linux scheduler), and the measuring application cyclictest ran on another core. This test ran uninterrupted for 24 hours. The graph below shows the result where the IRQ latency is in µs.

Figure 2. Result on Host



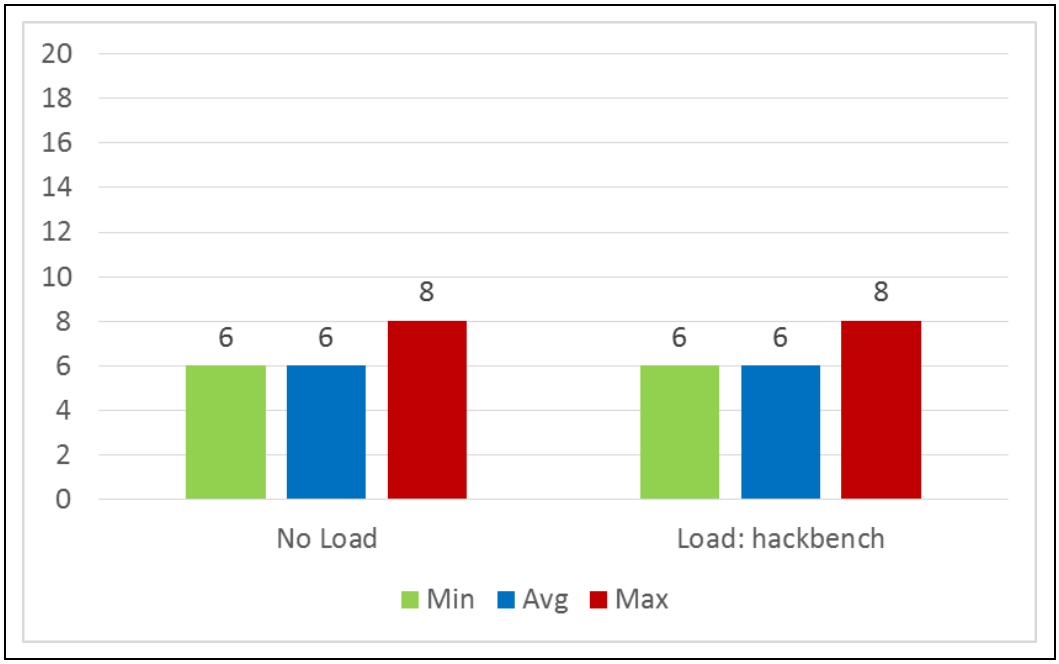
This result shows good consistency in the IRQ. An optimized system would have several spikes in the maximum IRQ latency. Further, the load on a core is not affecting the real-time task run on another core.

Note: The IRQ latency numbers on the host include the operating system, virtualization and real-time software overhead. The same in a bare-metal environment are significantly lower.

12.2 Result on VM

The host and guest OS has one dedicated core each for all generic activities. The load hackbench ran on one isolated core on the VM, and the measuring application cyclictest ran on another core on VM. This test ran uninterrupted for 24 hours. The graph below shows the result where the IRQ latency is in µs.

Figure 3. Result on VM



This result shows good consistency in the IRQ. The maximum latency is 8 µs for load and no load situation. Hence, it is safe to conclude that the load on another core of the VM is not affecting the real-time task run on another core.

§

13.0 Conclusion

A properly configured Intel® Xeon® server system can handle real-time tasks with great ease. The options suggested in the document should give the designers a better understanding on how to architect a real-time system using an Intel® Xeon® server.

In the IRQ latency test mentioned in the document, the IRQ latency in a VM running on an Intel® Xeon® server was under 10 µs. Since all the workloads do not need such a low IRQ latency, designers can selectively enable or disable certain options suggested in the document per their system design flexibility.

§