



Optimizing Linux for AMD EPYC with SUSE Linux Enterprise 12 SP3

SUSE Linux Enterprise Server 12 SP3

Mel Gorman, Senior Kernel Engineer, SUSE

Matt Fleming, Senior Performance Engineer, SUSE

Dario Faggioli, Software Engineer Virtualization Specialist, SUSE

Martin Jambor, Tool Chain Developer, SUSE

Brent Hollingsworth, Alliance Manager, AMD

The document at hand provides an overview of the AMD* EPYC* architecture and how computational-intensive workloads can be tuned on SUSE Linux Enterprise Server 12 SP3.

Publication Date: June 06, 2018

Contents

- 1 Overview 3
- 2 EPYC Architecture 3
- 3 EPYC Topology 4
- 4 Memory and CPU Binding 7
- 5 High-performance Storage Devices and Interrupt Affinity 10
- 6 Evaluating Workloads 11
- 7 Power Management 17
- 8 Security Mitigations 18
- 9 Hardware-based Profiling 19
- 10 Candidate Workloads 19
- 11 Using AMD EPYC for Virtualization 25
- 12 Conclusion 47
- 13 Resources 48
- 14 Glossary 48
- 15 Appendix A 49
- 16 Legal Notice 53
- 17 GNU Free Documentation License 54

1 Overview

EPYC is the latest generation of the AMD64 System-on-Chip (SoC) processor family. It is based on the Zen microarchitecture, introduced in 2017, and supports up to 32 cores (64 threads) and 8 memory channels per socket. At the time of writing, 1-socket and 2-socket models are available from Original Equipment Manufacturers (OEMs). This document provides an overview of the EPYC architecture and how computational-intensive workloads can be tuned on SUSE Linux Enterprise Server 12 SP3.

2 EPYC Architecture

Symmetric multiprocessing (SMP) systems are those that contain two or more physical processing cores. Each core may have two threads if hyper-threading is enabled, with some resources being shared between hyper-thread siblings. To minimize access latencies, multiple layers of caches are used, with each level being larger but with higher access costs. Cores may share different levels of cache which should be considered when tuning for a workload.

Historically, a single socket contained several cores sharing a hierarchy of caches and memory channels and multiple sockets were connected via a memory interconnect. Modern configurations may have multiple dies as a *Multi-Chip Module (MCM)* with one set of interconnects within the socket and a separate interconnect for each socket. This means that some CPUs and memory are faster to access than others depending on the “distance”. This should be considered when tuning for *Non-Uniform Memory Architecture (NUMA)* as all memory accesses are not necessarily to local memory incurring a variable access penalty.

EPYC is an MCM design with four dies on each package regardless of thread count. The number of cores on each die is always symmetric so they are balanced. Each socket has eight memory channels (two channels per die) with two *Dual Inline Memory Modules (DIMMs)* allowed per channel for up to 16 DIMMs per socket. Total capacity is expected to be 2 TB per socket with a maximum bandwidth of 21.3 GB/sec per channel for a total of 171 GB/sec per socket depending on the DIMMs selected.

Within the package, the four dies are interconnected with a fully-connected *Infinity Fabric*. Fully connected means that one core accessing memory connected to another die will always be one hop away. The bandwidth of the fabric is 42 GB/sec per link. The link is optimized for low-power and low-latency. Thus the bandwidth available means that a die accessing memory local to the socket incurs a smaller access penalty than is normally expected when accessing remote memory.

Sockets are also connected via Infinity Fabric with four links between each socket connecting each die on one socket to the peer die on the second socket. Consequently, access distance to remote memory from a thread will be at most two hops away. The data bandwidth on each of these links is 38 GB/sec for a total of 152 GB/sec between sockets. At the time of writing, only two sockets are possible within a single machine.

Power management on the links is careful to minimize the amount of power required. If the links are idle then the power may be used to boost the frequency of individual cores. Hence, minimizing access is not only important from a memory access latency point of view, but it also has an impact on the speed of individual cores.

There are two IO x16 links per die giving a total of 8 links where links can be used as Infinity links, PCI EXPRESS* links or a limited number of SATA* links. This allows very large IO configurations and a high degree of flexibility because of having a total of 128 lanes available on single socket machines. It is important to note that the number of links available is equal in one socket and two socket configurations. In one socket configurations, all lanes are available for IO. In two socket configurations, some lanes are used to connect the two sockets together with the upshot that a one socket configuration does not compromise on the available IO channels.

3 EPYC Topology

Figure 1 below shows the topology of an example machine generated by the `lstopo` tool.



FIGURE 1: EPYC TOPOLOGY

This tool is part of the `hwloc` package which is not supported in SUSE Linux Enterprise Server 12 SP3 but can be installed for illustration. The two “packages” correspond to each socket. The four dies on each socket are clearly visible and each die has a split L3 cache. Optimizing for computation should focus on co-operating tasks being bound to a die. In this example, the IO channels are not heavily used but the focus will be CPU and memory-intensive loads. If optimizing for IO, it is recommended, where possible, that the workload is located on the nodes local to the IO channel.

The computer output below shows a conventional view of the topology using the `numactl` tool. The CPU IDs that map to each node are reported on the “node X cpus:” lines and note the NUMA distances on the table at the bottom of the computer output. Node 0 and node 1 are a distance of 16 apart which is the distance between two dies on one socket. The distance between node 0 and node 4 is 32 as they are on separate sockets. The distance is a not a guarantee of the access latency. However, it is a rule of thumb that accesses between sockets are roughly twice the cost of accessing another die on the same socket.

```
epyc:~ # numactl --hardware
```

```

available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 64 65 66 67 68 69 70 71
node 0 size: 32056 MB
node 0 free: 31446 MB
node 1 cpus: 8 9 10 11 12 13 14 15 72 73 74 75 76 77 78 79
node 1 size: 32253 MB
node 1 free: 31545 MB
node 2 cpus: 16 17 18 19 20 21 22 23 80 81 82 83 84 85 86 87
node 2 size: 32253 MB
node 2 free: 31776 MB
node 3 cpus: 24 25 26 27 28 29 30 31 88 89 90 91 92 93 94 95
node 3 size: 32253 MB
node 3 free: 29039 MB
node 4 cpus: 32 33 34 35 36 37 38 39 96 97 98 99 100 101 102 103
node 4 size: 32253 MB
node 4 free: 31823 MB
node 5 cpus: 40 41 42 43 44 45 46 47 104 105 106 107 108 109 110 111
node 5 size: 32253 MB
node 5 free: 31565 MB
node 6 cpus: 48 49 50 51 52 53 54 55 112 113 114 115 116 117 118 119
node 6 size: 32253 MB
node 6 free: 32098 MB
node 7 cpus: 56 57 58 59 60 61 62 63 120 121 122 123 124 125 126 127
node 7 size: 32124 MB
node 7 free: 31984 MB
node distances:
node  0  1  2  3  4  5  6  7
 0:  10 16 16 16 32 32 32 32
 1:  16 10 16 16 32 32 32 32
 2:  16 16 10 16 32 32 32 32
 3:  16 16 16 10 32 32 32 32
 4:  32 32 32 32 10 16 16 16
 5:  32 32 32 32 16 10 16 16
 6:  32 32 32 32 16 16 10 16
 7:  32 32 32 32 16 16 16 10

```

Finally, the cache topology can be discovered in a variety of fashions. While `lstopo` can provide the information, it is not always available. Fortunately, the level, size and ID of CPUs that share cache can be identified from the files under `/sys/devices/system/cpu/cpuN/cache`.

4 Memory and CPU Binding

NUMA is a scalable memory architecture for multiprocessor systems that can reduce contention on a memory channel. A full discussion on tuning for NUMA is beyond the scope for this paper. But the document “A NUMA API for Linux” at <http://developer.amd.com/wordpress/media/2012/10/LibNUMA-WP-fv1.pdf> provides a valuable introduction.

The default policy for programs is the “local policy”. A program which calls `malloc()` or `mmap()` reserves virtual address space but does not immediately allocate physical memory. The physical memory is allocated the first time the address is accessed by any thread and, if possible, the memory will be local to the accessing CPU. If the mapping is of a file, the first access may have occurred at any time in the past so there are no guarantees about locality.

Memory allocated to a node is less likely to move if a thread changes to a CPU on another node or if multiple programs are remote accessing the data, unless *Automatic NUMA Balancing (NUMAB)* is enabled. When NUMAB is enabled, unbound process accesses are sampled. If there are enough remote accesses then the data will be migrated to local memory. This mechanism is not perfect and incurs overhead of its own. This means it can be important for performance for thread and process migrations between nodes to be minimized and for memory placement to be carefully considered and tuned.

The `taskset` tool is used to set or get the CPU affinity for new or existing processes. An example use is to confine a new process to CPUs local to one node. Where possible, local memory will be used. But if the total required memory is larger than the node then remote memory can still be used. In such configurations, it is recommended to size the workload such that it fits in the node. This avoids that any of the data is being paged out when `kswapd` wakes to reclaim memory from the local node.

`numactl` controls both memory and CPU policies for processes that it launches and can modify existing processes. In many respects, the parameters are easier to specify than `taskset`. For example, it can bind a task to all CPUs on a specified node instead of having to specify individual CPUs with `taskset`. Most importantly, it can set the memory allocation policy without requiring application awareness.

Using policies, a preferred node can be specified where the task will use that node if memory is available. This is typically used in combination with binding the task to CPUs on that node. If a workload's memory requirements are larger than a single node and predictable performance is required then the “interleave” policy will round-robin allocations from allowed nodes. This gives sub-optimal but predictable access latencies to main memory. More importantly, interleaving reduces the probability that the OS will need to reclaim any data belonging to a large task.

Further improvements can be made to access latencies by binding a workload to a single *CPU Complex (CCX)* within a node. Since L3 caches are not shared between CCXs, binding a workload to a CCX avoids L3 cache misses caused by workload migration.

Find examples below on how `taskset` and `numactl` can be used to start commands bound to different CPUs depending on the topology.

```
# Run a command bound to CPU 1
epyc:~ # taskset -c 1 [command]

# Run a command bound to CPUs belonging to node 0
epyc:~ # taskset -c `cat /sys/devices/system/node/node0/cpulist` [command]

# Run a command bound to CPUs belonging to nodes 0 and 1
epyc:~ # numactl -cpunodebind=0,1 [command]

# Run a command bound to CPUs that share L3 cache with cpu 1
epyc:~ # taskset -c `cat /sys/devices/system/cpu/cpu1/cache/index3/shared_cpu_list`
[command]
```

4.1 Tuning for Local Access Without Binding

The ability to use local memory where possible and remote memory if necessary is valuable but there are cases where it is imperative that local memory always be used. If this is the case, the first priority is to bind the task to that node. If that is not possible then the command `sysctl vm.zone_reclaim_mode=1` can be used to aggressively reclaim memory if local memory is not available.



Note: High Costs

While this option is good from a locality perspective, it can incur high costs because of stalls related to reclaim and the possibility that data from the task will be reclaimed. Treat this option with a high degree of caution and testing.

4.2 Hazards with CPU Binding

There are three major hazards to consider with CPU binding.

The first is to watch for remote memory nodes being used where the process is not allowed to run on CPUs local to that node. While going more in detail here is outside the scope of this paper, the most common scenario is an IO-bound thread communicating with a kernel IO thread on a remote node bound to the IO controller whose accesses are never local. Similarly, the version of `irqbalance` shipped with SUSE Linux Enterprise Server 12 SP3 is not necessarily optimal for EPYC. Thus it is worth considering disabling `irqbalance` and manually binding IRQs from storage or network devices to CPUs that are local to the IO channel. Depending on the kernel version and drivers in use, it may not be possible to manually bind IRQs. For example, some devices multi-queue support may not permit IRQs affinities to be changed.

The second is that guides about CPU binding tend to focus on binding to a single CPU. This is not always optimal when the task communicates with other threads as fixed bindings potentially miss an opportunity for the processes to use idle cores sharing an L1 or L2 cache. This is particularly true when dispatching IO, be it to disk or a network interface where a task may benefit from being able to migrate close to the related threads. But it also applies to pipeline-based communicating threads for a computational workload. Hence, focus *initially* on binding to CPUs sharing L3 cache and *then* consider whether to bind based on a L1/L2 cache or a single CPU using the primary metric of the workload to establish whether the tuning is appropriate.

The final hazard is similar in that if many tasks are bound to a smaller set of CPUs then the subset of CPUs could be over-saturated even though the machine overall has spare capacity.

4.3 CPUsets and Memory Control Groups

CPUsets are ideal when multiple workloads must be isolated on a machine in a predictable fashion. CPUsets allow a machine to be partitioned into subsets. These sets may overlap, and in that case they suffer from similar problems as CPU affinities. If there is no overlap, they can be switched to “exclusive” mode which treats them completely in isolation with relatively little overhead. The caveat in doing so is that one overloaded CPUset can be saturated leaving another CPUset completely idle. Similarly, they are well suited when a primary workload must be protected from interference because of low-priority tasks in which case the low priority tasks can be placed in a CPUset. The caveat with CPUsets is that the overhead is higher than using scheduler and memory policies. Ordinarily, the accounting code for CPUsets is completely disabled. But when a single CPUset is created there are additional essential checks that are made when checking scheduler and memory policies.

Similarly `memcg` can be used to limit the amount of memory that can be used by a set of processes. When the limits are exceeded then the memory will be reclaimed by tasks within `memcg` directly without interfering with any other tasks. This is ideal for ensuring there is no inference between two or more sets of tasks. Similar to CPUsets, there is some management overhead incurred so if tasks can simply be isolated on a NUMA boundary then it is preferred from a performance perspective. The major hazard is that if the limits are exceeded then the processes directly stall to reclaim the memory which can incur significant latencies.



Note:

Without `memcg`, when memory gets low, the global reclaim daemon does work in the background and if it reclaims quickly enough, no stalls are incurred. When using `memcg`, observe the `allocstall` counter in `/proc/vmstat` as this can detect early if stalling is a problem.

5 High-performance Storage Devices and Interrupt Affinity

High-performance storage devices like *Non-Volatile Memory Express (NVMe)* or *Serial Attached SCSI (SAS)* controller are designed to take advantage of parallel I/O submission. These devices typically support a large number of submit and receive queues, which are tied to *MSI-X* interrupts. Ideally these devices should provide as many *MSI-X* vectors as CPUs are present in the system. To achieve the best performance each *MSI-X* vector should be assigned to an individual CPU.

5.1 Automatic NUMA Balancing

Automatic NUMA Balancing will ignore any task that uses memory policies. If the workloads can be manually optimized with policies then do so and disable automatic NUMA balancing by specifying `numa_balancing=disable` on the kernel command line or via `sysctl`. There are many cases where it is impractical or impossible to specify policies in which case the balancing should be sufficient for throughput-sensitive workloads. For latency sensitive workloads, the sampling for NUMA balancing may be too high in which case it may be necessary to disable balancing. The final corner case where NUMA balancing is a hazard is a case where the number

of runnable tasks always exceeds the number of CPUs in a single node. In this case, the load balancer (and potentially affine wakes) will constantly pull tasks away from the preferred node as identified by automatic NUMA balancing resulting in excessive sampling and CPU migrations.

6 Evaluating Workloads

The first and foremost step when evaluating how a workload should be tuned is to establish a primary metric such as latency, throughput or elapsed time. When each tuning step is considered or applied, it is critical that the primary metric be examined before conducting any further analysis to avoid intensive focus on the wrong bottleneck. Make sure that the metric is measured multiple times to ensure that the result is reproducible and reliable within reasonable boundaries. When that is established, analyse how the workload is using different system resources to determine what area should be the focus. The focus in this paper is on how CPU and memory is used. But other evaluations may need to consider the IO subsystem, network subsystem, system call interfaces, external libraries etc. The methodologies that can be employed to conduct this are outside the scope of the paper but the book “Systems Performance: Enterprise and the Cloud” by Brendan Gregg (see <http://www.brendangregg.com/sysperfbok.html>) is a recommended primer on the subject.

6.1 CPU Utilization and Saturation

Decisions on whether to bind a workload to a subset of CPUs require that the CPU utilization and any saturation risk is known. Both the `ps` and `pidstat` commands can be used to sample the number of threads in a system. Typically `pidstat` yields more useful information with the important exception of the run state. A system may have many threads but if they are idle then they are not contributing to utilization. The `mpstat` command can report the utilization of each CPU in the system.

High utilization of a small subset of CPUs may be indicative of a single-threaded workload that is pushing the CPU to the limits and may indicate a bottleneck. Conversely, low utilization may indicate a task that is not CPU-bound, is idling frequently or is migrating excessively. While each workload is different, load utilization of CPUs may show a workload that can run on a subset of CPUs to reduce latencies because of either migrations or remote accesses. When utilization is high, it is important to determine if the system could be saturated. The `vmstat` tool reports the number of runnable tasks waiting for CPU in the “r” column where any value over 1 indicates

that wakeup latencies may be incurred. While the exact wakeup latency can be calculated using trace points, knowing that there are tasks queued is an important step. If a system is saturated, it may be possible to tune the workload to use fewer threads.

Overall, the initial intent should be to use CPUs from as few NUMA nodes as possible to reduce access latency but there are exceptions. EPYC has an exceptional number of high-speed memory channels to main memory, thus consider the workload thread activity. If they are co-operating threads or sharing data then isolate them on as few nodes as possible to minimize cross-node memory accesses. If the threads are completely independent with no shared data, it may be best to isolate them on a subset of CPUs from each node to maximize the number of available memory channels and throughput to main memory. For some computational workloads, it may be possible to use hybrid models such as MPI for parallelization across nodes and using openMP for threads within nodes.

6.2 Transparent Huge Pages

Huge pages are a mechanism by which performance can be improved. This happens by reducing the number of page faults, the cost of translating virtual addresses to physical addresses because of fewer layers in the page table and by being able to cache translations for a larger portion of memory. *Transparent Huge Pages (THP)* is supported for private anonymous memory that automatically backs mappings with huge pages where anonymous memory could be allocated as `heap`, `malloc()`, `mmap(MAP_ANONYMOUS)`, etc. While the feature has existed for a long time, it has evolved significantly.

Many tuning guides recommend disabling THP because of problems with early implementations. Specifically, when the machine was running for long enough, the use of THP could incur severe latencies and could aggressively reclaim memory in certain circumstances. These problems have been resolved by the time SUSE Linux Enterprise Server 12 SP3 was released. This means there are no good grounds for automatically disabling THP because of severe latency issues without measuring the impact. However, there are exceptions that may be considered for specific workloads.

Some high-end in-memory databases and other applications aggressively use `mprotect()` to ensure that unprivileged data is never leaked. If these protections are at the base page granularity then there may be many THP splits and rebuilds that incur overhead. It can be identified if this is a potential problem by using `strace` to detect the frequency and granularity of the system call. If they are high frequency then consider disabling THP. It can also be sometimes inferred from observing the `thp_split` and `thp_collapse_alloc counters` in `/proc/vmstat`.

Workloads that sparsely address large mappings may have a higher memory footprint when using THP. This could result in premature reclaim or fallback to remote nodes. An example would be HPC workloads operating on large sparse matrices. If memory usage is much higher than expected then compare memory usage with and without THP to decide if the tradeoff is not worthwhile. This may be critical on EPYC given that any spillover will congest the Infinity links and potentially cause cores to run at a lower frequency.



Note: Sparsely Addressed Memory

This is specific to sparsely addressed memory. A secondary hint for this case may be that the application primarily uses large mappings with a much higher Virtual Size (VSZ, see [Section 6.1, “CPU Utilization and Saturation”](#)) than Resident Set Size (RSS). Applications which densely address memory benefit from the use of THP by achieving greater bandwidth to memory.

Parallelized workloads that operate on shared buffers with threads using more CPUs that are on a single node may experience a slowdown with THP if the granularity of partitioning is not aligned to the huge page. The problem is that if a large shared buffer is partitioned on a 4K boundary then false sharing may occur whereby one thread accesses a huge page locally and other threads access it remotely. If this situation is encountered, it is preferable that the granularity of sharing is increased to the THP size. But if that is not possible then disabling THP is an option.

Applications that are extremely latency sensitive or must always perform in a deterministic fashion can be hindered by THP. While there are fewer faults, the time for each fault is higher as memory must be allocated and cleared before being visible. The increase in fault times may be in the microsecond granularity. Ensure this is a relevant problem as it typically only applies to hard real-time applications. The secondary problem is that a kernel daemon periodically scans a process looking for contiguous regions that can be backed by huge pages. When creating a huge page, there is a window during which that memory cannot be accessed by the application and new mappings cannot be created until the operation is complete. This can be identified as a problem with thread-intensive applications that frequently allocate memory. In this case consider effectively disabling `khugepaged` by setting a large value in `/sys/kernel/mm/transparent_hugepage/khugepaged/alloc_sleep_millisecs`. This will still allow THP to be used opportunistically while avoiding stalls when calling `malloc()` or `mmap()`.

THP can be disabled. To do so, specify `transparent_hugepage=disable` on the kernel command line, at runtime via `/sys/kernel/mm/transparent_hugepage/enabled` or on a per process basis by using a wrapper to execute the workload that calls `prctl(PR_SET_THP_DISABLE)`.

6.3 User/Kernel Footprint

Assuming an application is mostly CPU or memory bound, it is useful to determine if the footprint is primarily in user space or kernel space. The reason for this is that it gives a hint where tuning should be focused. The percentage of CPU time can be measured on a coarse-grained fashion using `vmstat` or a fine-grained fashion using `mpstat`. If an application is mostly spending time in user space then the focus should be on tuning the application itself. If the application is spending time in the kernel then it should be determined which subsystem dominates. The `strace` or `perf trace` commands can measure the type, frequency and duration of system calls as they are the primary reasons an application spends time within the kernel. In some cases, an application may be tuned or modified to reduce the frequency and duration of system calls. In other cases, a profile is required to identify which portions of the kernel are most relevant as a target for tuning.

6.4 Memory Utilization and Saturation

The traditional means of measuring memory utilization of a workload is to examine the *Virtual Size (VSZ)* and *Resident Set Size (RSS)* using either the `ps` or `pidstat` tool. This is a reasonable first step but is potentially misleading when shared memory is used and multiple processes are examined. VSZ is simply a measure of memory space reservation and is not necessarily used. RSS may be double accounted if it is a shared segment between multiple processes. The file `/proc/pid/maps` can be used to identify all segments used and whether they are private or shared. The file `/proc/pid/smmaps` yields more detailed information including the *Proportional Set Size (PSS)*. PSS is an estimate of RSS except it is divided between the number of processes mapping that segment which can give a more accurate estimate of utilization. Note that the `smmaps` file is very expensive to read and should not be monitored at a high frequency. Finally, the *Working Set Size (WSS)* is the amount of memory active required to complete computations during an arbitrary phase of a programs execution. It is not a value that can be trivially measured. But conceptually it is useful as the interaction between WSS relative to available memory affects memory residency and page fault rates.

On NUMA systems, the first saturation point is a node overflow when the “local” policy is in effect. Given no binding of memory, when a node is filled, a remote node’s memory will be used transparently and background reclaim will take place on the local node. Two consequences of this are that remote access penalties will be used and old memory from the local node will be reclaimed. If the WSS of the application exceeds the size of a local node then paging and refaults may be incurred.

The first thing to identify is that a remote node overflow occurred which is accounted for in `/proc/vmstat` as the `numa_hit`, `numa_miss`, `numa_foreign`, `numa_interleave`, `numa_local` and `numa_other` counters:

- `numa_hit` is incremented when an allocation uses the preferred node where preferred may be either a local node or one specified by a memory policy.
- `numa_miss` is incremented when an alternative node is used to satisfy an allocation.
- `numa_foreign` is rarely useful but is accounted against a node that was preferred. It is a subtle distinction from `numa_miss` that is rarely useful.
- `numa_interleave` is incremented when an interleave policy was used to select allowed nodes in a round-robin fashion.
- `numa_local` increments when a local node is used for an allocation regardless of policy.
- `numa_other` is used when a remote node is used for an allocation regardless of policy.

For the local memory policy, the `numa_hit` and `numa_miss` counters are the most important to pay attention to. An application that is allocating memory that starts incrementing the `numa_miss` implies that the first level of saturation has been reached. If this is observed on EPYC, it may be valuable to bind the application to nodes that represent dies on a single socket. If the ratio of hits to misses is close to 1, consider an evaluation of the interleave policy to avoid unnecessary reclaim.



Note: NUMA Statistics

These NUMA statistics only apply at the time a physical page was allocated and it is not related to the reference behaviour of the workload. For example, if a task running on node 0 allocates memory local to node 0 then it will be accounted for as a `node_hit` in the statistics. However, if the memory is shared with a task running on node 1, all the

accesses may be remote, which is a miss from the perspective of the hardware but not accounted for in `/proc/vmstat`. Detecting remote and local accesses at a hardware level requires using the hardware's *Performance Management Unit*.

When the first saturation point is reached then reclaim will be active. This can be observed by monitoring the `pgscan_kswapd` and `pgsteal_kswapd` `/proc/vmstat` counters. If this is matched with an increase in major faults or minor faults then it may be indicative of severe thrashing. In this case the interleave policy should be considered. An ideal tuning option is to identify if shared memory is the source of the usage. If this is the case, then interleave the shared memory segments. This can be done in some circumstances using `numactl` or by modifying the application directly.

More severe saturation is observed if the `pgscan_direct` and `pgsteal_direct` counters are also increasing as these indicate that the application is stalling while memory is being reclaimed. If the application was bound to individual nodes, increasing the number of available nodes will alleviate the pressure. If the application is unbound, it indicates that the WSS of the workload exceeds all available memory. It can only be alleviated by tuning the application to use less memory or increasing the amount of RAM available.

As before, whether to use memory nodes from one socket or two sockets depends on the application. If the individual processes are independent then either socket can be used. But where possible, keep communicating processes on the same socket to maximize memory throughput while minimizing the socket interconnect traffic.

6.5 Other Resources

The analysis of other resources is outside the scope of this paper. However, a common scenario is that an application is IO-bound. A superficial check can be made using the `vmstat` tool and checking what percentage of CPU time is spent idle combined with the number of processes that are blocked and the values in the `bi` and `bo` columns. Further analysis is required to determine if an application is IO rather than CPU or memory bound. But this is a sufficient check to start with.

7 Power Management

Modern CPUs balance power consumption and performance through *Performance States (P-States)*. Low utilization workloads may use lower P-States to conserve power while still achieving acceptable performance. When a CPU is idle, lower power idle states (*C-States*) can be selected to further conserve power. However this comes with higher exit latencies when lower power states are selected. It is further complicated by the fact that if individual cores are idle and running at low power then the additional power can be used to boost the performance of active cores. This means this scenario is not a straight-forward balance between power consumption and performance. More complexity is added on EPYC whereby spare power may be used to boost either cores or the Infinity links.

EPYC provides *SenseMI* which, among other capabilities, enables CPUs to make adjustments to voltage and frequency depending on the historical state of the CPU. There is a latency penalty when switching P-States but EPYC is capable of fine-grained in the adjustments that can be made to reduce likelihood that the latency is a bottleneck. On SUSE Linux Enterprise Server, EPYC uses the `acpi_cpufreq` driver which allows P-states to be configured to match requested performance. However, this is limited in terms of the full capabilities of the hardware. It cannot boost the frequency beyond the maximum stated frequencies and if a target is specified then the highest frequency below the target will be used. A special case is if the governor is set to performance. In this situation the hardware will quickly use the highest available frequency in an attempt to work quickly and then return to idle.

What should be determined is whether power management is likely to be a factor for a workload. One that is limited to a subset of active CPUs and nodes will have high enough utilization so that power management will not be active on those cores and no action is required. Hence, with CPU binding, the issue of power management may be side-stepped.

Secondly, a workload that does not communicate heavily with other processes and is mostly CPU-bound will also not experience any side effects because of power management.

The workloads that are most likely to be affected are those that synchronously communicate between multiple threads or those that idle frequently and have low CPU utilization overall. It will be further compounded if the threads are sensitive to wakeup latency but there are secondary effects if a workload must complete quickly but the CPU is running at a low frequency. The P-State and C-State of each CPU can be examined using the `turbostat` utility. The computer output below shows an example where one workload is busy on CPU 0 and other workloads are idle. A useful exercise is to start a workload and monitor the output of `turbostat` paying

close attention to CPUs that have moderate utilization and running at a lower frequency. If the workload is latency-sensitive then it is grounds for either minimizing the number of CPUs available to the workload or configuring power management.

Package	Core	CPU	Avg_MHz	Busy%	Bzy_MHz	TSC_MHz	IRQ	C1	C2	C1%	C2%
-	-	-	26	0.85	3029	2196	5623	1251	3439	0.64	98.53
0	0	0	3192	100.00	3192	2196	1268	0	0	0.00	0.00
0	0	64	1	0.02	2936	2196	10	0	9	0.00	99.99
0	1	1	1	0.08	1337	2196	20	0	14	0.00	99.94
0	1	65	1	0.04	1263	2196	13	0	12	0.00	99.97
0	2	2	1	0.04	1236	2196	14	0	12	0.00	99.98
0	2	66	1	0.04	1226	2196	14	0	13	0.00	99.97
0	3	3	1	0.05	1237	2196	16	0	14	0.00	99.97
0	3	67	1	0.05	1238	2196	16	0	15	0.00	99.97

In the event it is determined that tuning CPU frequency management is appropriate. Then the following actions can be taken to set the management policy to performance using the `cpupower` utility:

```
epyc:~# cpupower frequency-set -g performance
Setting cpu: 0
Setting cpu: 1
Setting cpu: 2
...
```

Persisting it across reboots can be done via a local `init` script, via `udev` or via one-shot `systemd` service file if it is deemed to be necessary. Note that `turbostat` will still show that idling CPUs use a low frequency. The impact of the policy is that the highest P-State will be used as soon as possible when the CPU is active. In some cases, a latency bottleneck will occur because of a CPU exiting idle. If this is identified on EPYC, restrict the C-state by specifying `processor.max_cstate=2` on the kernel command line which will prevent CPUs from entering lower C-states. It is expected on EPYC that the exit latency from C1 is very low. But by allowing C2, it reduces interference from the idle loop injecting micro-operations into the pipeline and should be the best state overall. It is also possible to set the max idle state on individual cores using `cpupower idle-set`. If SMT is enabled, the idle state should be set on both siblings.

8 Security Mitigations

On occasion, a security fix is applied to a distribution that has a performance impact. The most recent notable examples are **Meltdown** and two variants of **Spectre**. AMD EPYC is immune to the Meltdown variant and page table isolation is never active. However, it is vulnerable to the

Spectre variant. In the event it can be guaranteed that the server is in a trusted environment running only known code that is not malicious, the `nospectre_v2` parameter can be specified on the kernel command line. This is only relevant to workloads that enter/exit the kernel frequently.

9 Hardware-based Profiling

Ordinarily advanced monitoring of a workload is conducted via `oprofile` or `perf`. At the time of writing, it is known that EPYC has extensive Performance Monitoring Unit (PMU) capabilities but the OS support is limited. `oprofile` is not implemented and falls back to using the timer interrupt which is not recommended for general use. `perf` support is limited to a subset of events: cycles, L1 cache access/misses, TLB access/misses, retired branch instructions and mispredicted branches. In terms of identifying what subsystem may be worth tuning in the OS, the most useful invocation is `perf record -a -e cycles sleep 30` to capture 30 seconds of data for the entire system. You can also call `perf record -e cycles command` to gather a profile of a given workload. Specific information on the OS can be gathered through tracepoints or creating probe points with `perf` or `trace-cmd`. But the details on how to conduct such analysis are beyond the scope of this paper.

10 Candidate Workloads

The workloads that will benefit most from the EPYC architecture are those that can be parallelized and are either memory or IO-bound. This is particularly true for workloads that are “NUMA friendly”: they can be trivially parallelized and each thread can operate independently for the majority of the workloads lifetime. For memory-bound workloads, the primary benefit will be taking advantage of the high bandwidth available on each channel. For IO-bound workloads, the primary benefit will be realized when there are multiple storage devices, each of which is connected to the node local to a task issuing IO.

10.1 Test Setup

The following sections will demonstrate how an OpenMP and MPI workload can be configured and tuned on an EPYC reference platform.

TABLE 1: TEST SETUP

CPU	2x AMD EPYC 7601
Platform	AMD Speedway Reference Platform
Drive	Samsung SSD 850
OS	SUSE Linux Enterprise Server 12 SP3
Memory Interleaving	Channel
Memory Speed	2400MHz (single rank)
Kernel command line	<u>nospectre_v2</u>

10.2 Test workload: STREAM

STREAM is a memory bandwidth benchmark created by Dr. John D. McCalpin from the University of Virginia (for more information, see <https://www.cs.virginia.edu/stream/>). It can be used to measure bandwidth of each cache level and bandwidth to main memory assuming adequate care is taken. It is not perfect as some portions of the data will be stored in cache instead of being fetched from main memory.

The benchmark was configured to run both single-threaded and parallelised with OpenMP to take advantage of each memory channel. The array elements for the benchmark was set at 100007936 elements at compile time so each that array was 763MB in size for a total memory footprint of 2289 MB. The size was selected to minimize the possibility that cache usage would dominate the measurements.

TABLE 2: TEST WORKLOAD: STREAM

Compiler	gcc (SUSE Linux Enterprise) 4.8.5
Compiler flags	<u>-m64 -lm -O3</u>
OpenMP compiler flag	<u>-fopenmp</u>
OpenMP environment variables	<u>OMP_PROC_BIND=SPREAD</u> <u>OMP_NUM_THREADS=16</u>

The number of openMP threads was selected to have at least one thread running for every memory channel. The `OMP_PROC_BIND` parameter was to have one thread running on a core with a dedicated L3 cache to maximize available bandwidth. This can be verified using `trace-cmd`, as illustrated below with slight editing for formatting and clarity.

```
epyc:~ # trace-cmd record -e sched:sched_migrate_task ./stream
epyc:~ # trace-cmd report
...
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18799 prio=120 orig_cpu=0
dest_cpu=4
stream-18798 [000] x: sched_migrate_task: comm=trace-cmd pid=18670 prio=120 orig_cpu=4
dest_cpu=5
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18800 prio=120 orig_cpu=0
dest_cpu=8
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18801 prio=120 orig_cpu=0
dest_cpu=12
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18802 prio=120 orig_cpu=0
dest_cpu=16
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18803 prio=120 orig_cpu=0
dest_cpu=20
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18804 prio=120 orig_cpu=0
dest_cpu=24
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18805 prio=120 orig_cpu=0
dest_cpu=28
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18806 prio=120 orig_cpu=0
dest_cpu=32
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18807 prio=120 orig_cpu=0
dest_cpu=36
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18808 prio=120 orig_cpu=0
dest_cpu=40
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18809 prio=120 orig_cpu=0
dest_cpu=44
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18810 prio=120 orig_cpu=0
dest_cpu=48
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18811 prio=120 orig_cpu=0
dest_cpu=52
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18812 prio=120 orig_cpu=0
dest_cpu=56
stream-18798 [000] x: sched_migrate_task: comm=stream pid=18813 prio=120 orig_cpu=0
dest_cpu=60
```

Figure 2 below shows the reported bandwidth for the single and parallelized case. The single-threaded bandwidth for a single core was roughly 20 GB/sec which is a high percentage of the theoretical max of 38.4 GB/sec (each core has access to two memory channels). The channels are interleaved in this configuration as it has been recommended as the best balance for a

variety of workloads but limits the absolute maximum of a specialized benchmark like STREAM. The total throughput for each parallel operation ranged from 174 GB/sec to 240 GB/sec which is comparable to the theoretical maximum of 307 GB/sec.



Note: STREAM Scores

Higher STREAM scores can be reported by reducing the array sizes so that cache is partially used with the maximum score requiring that each threads memory footprint fits inside the L1 cache. Additionally, it is possible to achieve results closer to the theoretical maximum by manual optimization of the STREAM benchmark using vectored instructions and explicit scheduling of loads and stores. The purpose of this configuration was to illustrate the impact of properly binding a workload that can be fully parallelized with data-independent threads.

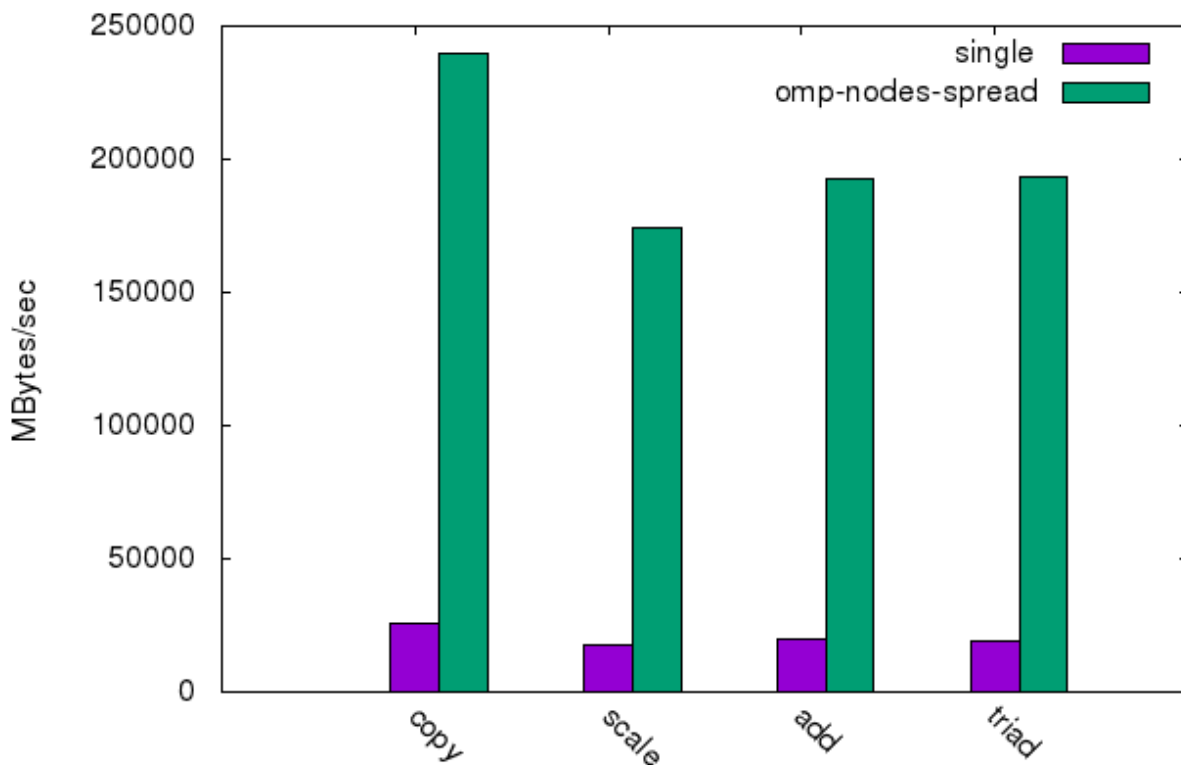


FIGURE 2: STREAM BANDWIDTH, SINGLE THREADED AND PARALLELIZED

10.3 Test Workload: NASA Parallel Benchmark

NASA Parallel Benchmark (NPB) is a small set of programs designed to evaluate the performance of supercomputers. They are small kernels derived from *Computational Fluid Dynamics (CFD)* applications. The problem size can be adjusted for different memory sizes. Reference implementations exist for both MPI and OpenMP. This setup will focus on the MPI reference implementation. While each application behaves differently, one common characteristic is that the workload is very context-switch intensive, barriers frequently yield the CPU to other tasks and the lifetime of individual processes can be very short-lived. The following paragraphs detail the tuning selected for this workload.

The most important step is setting the CPU governor to “performance”. This needs to be done because of the short-lived nature of some tasks but also because not all of them run long enough for a higher P-State to be selected even though the workload is very throughput sensitive. The migration cost parameter is set to reduce the frequency the load balancer will move an individual task. The minimum granularity is adjusted to reduce over-scheduling effects.



Note: Number of MPI Processes

Only 64 MPI processes were used for this test workload even though more CPUs are available.

This particular workload requires a power-of-two number of processes to be used but using all available CPUs means that the application can contend with itself for CPU time. Furthermore, as IO is being issued to shared memory backed by disk, there are system threads that also need CPU time. Finally, binding to the L3 cache means that there were more MPI worker processes than there are CPUs available that share a cache. If more threads were to be used, it would be necessary to bind on a per-node basis. As NPB uses shared files, an XFS partition was used for the temporary files albeit it is only used for mapping shared files and is not a critical path for the benchmark and no IO tuning is necessary. In some cases with MPI applications, it will be possible to use a `tmpfs` partition for OpenMPI. This avoids unnecessary IO assuming the increased physical memory usage does not cause the application to be paged out.

TABLE 3: TEST WORKLOAD: NASA PARALLEL BENCHMARK

Compiler	gcc (SUSE Linux Enterprise) 4.8.5, mpif77, mpicc
OpenMPI	openmpi 1.10.6-3.3.5SDYC14159b

Compiler flags	<code>-m64 -O2 -mcmodel=large</code>
CPU governor performance	<code>cpupower frequency-set -g performance</code>
Scheduler parameters	<code>sysctl -w kernel.sched_migration_cost_ns=5000000</code> <code>sysctl -w kernel.sched_min_granularity_ns=10000000</code>
mpirun parameters	<code>-mca btl ^openib,udapl -np 64 --bind-to l3cache</code>
mpirun environment	<code>TMPDIR=/xfs-data-partition</code>

Figure 3 shows the time, as reported by the benchmark, for each of the kernels to complete.

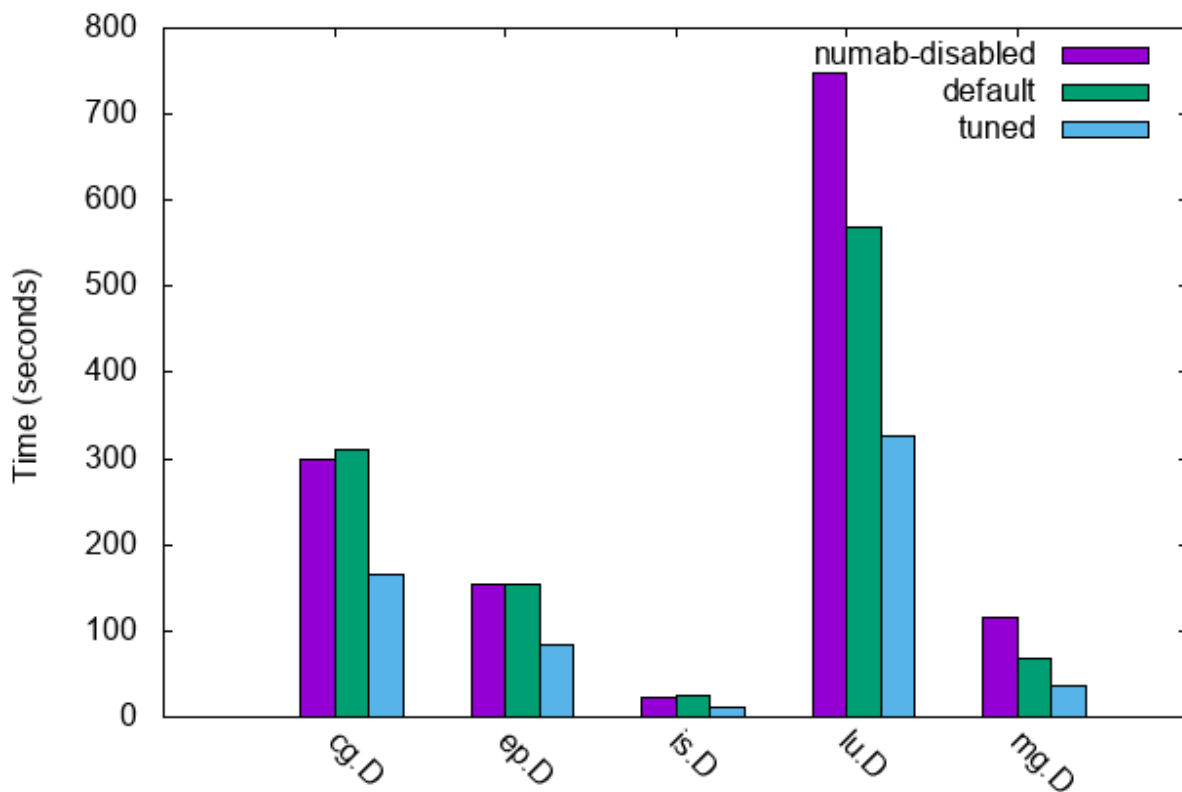


FIGURE 3: NAS MPI RESULTS

The baseline is “numab-disabled” which means it has disabled Automatic NUMA Balancing. The second test was a default SUSE Linux Enterprise Server 12 SP3 installation with no tuning. It illustrates that even with the overhead of Automatic NUMA Balancing there are savings overall as most of the workloads complete faster. The final test run has the tuning applied and shows

that the workload completes 44% to 68% faster than the baseline. When the workload is tuned with the bindings then Automatic NUMA Balancing can be optionally disabled but the difference in performance is marginal.

11 Using AMD EPYC for Virtualization

On first approximation, Virtual Machines (VMs) can be considered as large (in terms of memory footprint) and long running applications. Thus the tuning described so far in the paper can be applied.

However, when taking into account more specific aspects and characteristics of VMs, and making specific considerations about virtualization, a better tailored and more effective set of tuning advice can be derived. This is especially relevant for NUMA systems, such as AMD EPYC:

- VMs are long running activities, and typically use much more memory than “regular” OS processes.
- VMs can be configured to be, and act, both like NUMA-aware and non NUMA-aware workloads.

Calling VMs “long running activities” means that they often run for hours, days, or even months, without being terminated or restarted. Therefore, it is almost never acceptable to pay the price of suboptimal resource partitioning and allocation, even when there is the expectation that things will be better next time. Poor mapping of virtual machine resources (virtual CPUs and memory, but also I/O) on the host topology may cause issues to everything that runs inside the virtual machine – and potentially even to other components of the system – for a long time.

With reference to NUMA-awareness, a VM is called out to be NUMA aware, if a (virtual) NUMA topology is defined and exposed to the VM itself, and if the OS that the VM runs (guest OS) is NUMA-aware. On the contrary, a VM is called NUMA-unaware, if either no (virtual) NUMA topology is exposed, or the guest OS is not NUMA-aware.

In general, VMs that are large enough (in terms of amount of memory and number of virtual CPUs) to span multiple host NUMA nodes, benefit from being configured as NUMA-aware VMs. However, even for small and NUMA-unaware VMs, intelligent placement of their memory on the host nodes, and effective mapping of their virtual CPUs (vCPUs) on the host physical CPUs (pCPUs) is key for achieving good and consistent performance.

The following sections of this paper focuses on tuning for CPU and memory intensive VMs, and leave IO aside. More specifically, it focuses on how to configure and tune one or more VMs, so that CPU and memory intensive workloads running inside them can achieve the best performance.

It is highly desirable that vCPUs run close to the memory that they are accessing (for example on the same node). For reasonably big NUMA-aware VMs that requires properly mapping the virtual NUMA nodes of the guest to physical NUMA nodes on the host. For smaller NUMA-unaware VMs that means allocating all their memory on the smallest possible number of host NUMA nodes (better if just one), and making their vCPUs run on the pCPUs of those nodes respectively that node.

Both the Kernel-based Virtual Machine (KVM) and the Xen-Project hypervisors, as they are available in SUSE Linux Enterprise Server 12 SP3, provide (slightly different) mechanisms to enact this kind of resource partitioning and allocation.

11.1 Preparing the Host for Virtualization

Giving details on how to install and configure a system, so that it becomes a suitable virtualization host, is outside of the scope of this paper. For instructions and details refer to the SUSE documentation at https://www.suse.com/documentation/sles-12/book_virt/data/cha_vt_installation.html .

The same applies to configuring both the system's and the VMs' networking and storage. For specific details refer to the operating system, libvirt or hypervisor documentation and manuals. For example, to know how to assign network interfaces (or ports) to one or more VMs for improved network performance, refer to the SUSE documentation at https://www.suse.com/documentation/sles-12/book_virt/data/sec_libvirt_config_pci.html .

11.2 Virtual Machine Types

KVM only supports one type of VM – a fully hardware-based virtual machine (HVM). Under Xen, VMs can be paravirtualized (PV) or hardware virtualized machines (HVM). Xen also supports mixed modes. For example, hardware virtualized VMs can use some paravirtualized interfaces.

Xen HVM guests with paravirtualized interfaces enabled (often called PVHVM, or for brevity, HVM) are very similar to KVM VMs (which also use both hardware virtualization and paravirtualized IO, namely virtIO). This paper is always referring to PVHVM VMs when talking about VMs running on Xen.

11.3 Oversubscription of Host Resources

Oversubscription happens when the demand for some resource is higher than is physically available. In virtualization, this is typical for vCPUs, and can also happen for memory.



Note: Not Covering Oversubscribed Scenarios

Given the large number of CPUs that can be available on an EPYC system (128 in the example in *Figure 1, "EPYC Topology"*), and the huge amount of memory the architecture supports, not covering oversubscribed scenarios is not considered a limitation, at least as far as this paper is concerned.

In any case, most of the tuning that will be illustrated here is valid for oversubscribed systems as well. VM configuration advises can easily be adapted to be effective in such a scenario.

CPU Oversubscription

CPU oversubscription is what happens when, on a 128 physical CPUs system, the administrator creates, for example, 200 single vCPU guests. It is impossible to say whether this configuration is good, and should be encouraged, or bad, and should be avoided or forbidden, without further knowledge about the actual goals of the system itself, and – even more important – about the workloads.

As an example, if the load on each vCPU will always stay below 50 percent, oversubscribing by a factor of 2 would not only be tolerated, but would be advisable to avoid wasting resources. In this scenario that means creating 256 single vCPU VMs on the 128 pCPUs host is a good configuration (this, for simplicity, does not take into account the host OS, which will be discussed later). On the contrary, if it is known that the load on each vCPU will always be 100 percent, creating even 129 single vCPU VMs is already a misconfiguration (although it would likely be tolerated and handled well).

After all, hypervisors have schedulers to deal correctly with situations when there are more runnable entities (that is, vCPUs) than there is capacity to actually execute them at the same time (that is, pCPUs). The benefit of running more workloads (that is, VMs) than the hardware would allow comes at the price of reduced throughput and increased latency for the workloads themselves. As the focus of this paper is mainly on CPU and memory intensive workloads, which fully load the vCPUs on which they run than on IO bound ones, CPU oversubscription is out of scope here and only briefly mentioned.

Memory Oversubscription

There are a few ways of achieving memory oversubscription. The first, which we can call “classical memory oversubscription”, is what happens when an administrator creates VMs with a total cumulative memory footprint greater than the amount of physical RAM on the host. This only works if some of such memory is kept outside of the RAM (*swapped out*) when the VMs using it are not running and put back inside of the RAM (*swapped in*) when they are. Oversubscribing memory on KVM only requires creating VMs whose total amount of memory exceeds the host’s RAM. The usual Linux kernel virtual memory management and paging mechanisms will be used to handle that. On Xen, this variant of memory oversubscription is not possible unless special technologies (for example, the [xenpaging](#) tool and/or transcendent memory) are employed.

Another way of doing memory oversubscription is page sharing or page merging. This is based on the principle that if two (or more) VMs happen to use two (or more) pages, the content of which is identical, it would be enough to keep one in memory and only refer to it from the other places. Similar to overcommitting via paging, this is available natively on KVM via a mechanism called *Kernel Samepage Merging (KSM)*. On Xen, it needs special actions.

Finally, there is memory ballooning. This concept is based on the fact that VMs may not need all the memory they are given by the system administrator all the time. This means, although a VM will appear to always have all its memory, some of that memory is not actually allocated onto the host RAM (ballooned down) until the VM actually uses it (ballooning up). This is supported in both Xen and KVM.

Whichever method is used, allowing memory oversubscription has both latency and throughput implications. This makes it unideal for the workloads considered in this paper, and it is therefore not further explored here.

Oversubscription with a single VM

In the case where only one VM is configured on the host,

- a single VM should never have more vCPUs than the host has pCPUs.
- a single VM should never have more memory than the host has physical RAM.

11.4 Resource Allocation and Tuning of the Host

The main purpose of a system being used as a virtualization host is running VMs. To do that effectively, there are activities which reside and run on the host Operating System (host OS). These processes require some resources and are subject to being tuned. In fact, on both Xen and KVM, the host OS is at least responsible for helping with the IO performed by the VMs.

11.4.1 Allocating Resources to the Host OS

It is generally recognized as a good practise to make sure that the host OS has some resources assigned to itself. This may mean that some physical CPUs, and some memory, will be exclusively granted to the host OS.



Note: Host OS on KVM and on Xen

While on KVM the host OS is the actual Linux operating system that loads the hypervisor kernel modules, on Xen the host OS runs inside what is effectively a very special guest VM.

It is hard to give general recommendations but a rule of thumb (validated by other performance tuning efforts) suggests that 5 percent to 10 percent of the physical RAM should be assigned to the host OS. Even more, in case the plan is to run hundreds of VMs. However, if using Xen, that can be reduced to a few GigaBytes, even when planning to spawn many VMs. This is especially true if disaggregation is used (see https://wiki.xenproject.org/wiki/Dom0_Disaggregation).

In terms of CPUs, on “traditional NUMA systems”, where NUMA nodes correspond to sockets, it is usually advised to reserve one physical core (which means two logical CPUs, considering hyperthreading) per socket for the host OS. This translates to one physical core per node on EPYC. Usually it will be fine to assign less CPUs than that to the host OS, but it is better to always give one core to it, for each node that has IO channels attached. Host OS activity is mostly related to performing IO, and the kernel threads dedicated to the handling of actual devices, which are

often bound to the nodes on which the devices are attached, are better when given good chances to run without much contention. In the example architecture shown in *Figure 1, "EPYC Topology"* this would mean reserving one physical core for the host OS on nodes 0, 1, 3, 4 and 5.

As system administrators need to be able to reach out and login to the system, to manage and troubleshoot it, some resources should be reserved for management consoles, and the chosen hypervisor's toolstack (for example, the **SSH** daemon and the **libvirt** daemon).

All that has been described so far is greatly workload dependant. Since each VM does some IO, the ideal setup would be to dedicate one host physical or logical core to doing IO for each device used by each VM (or at least for those devices important for the specific workload, for example, network IO, for VMs doing network intensive activities). But this reduces the number of CPUs available for running VMs, which may be a problem. Also, considering that this grants sensible performance improvements only to IO intensive workloads (which are outside of the scope of this paper anyway), it may not be considered worthwhile.

If the overall goal of the system is running one or two big VMs, for example on a host like the one in *Figure 1, "EPYC Topology"*, with 128 (logical) CPUs – and each VM as 4 IO devices – it is sufficient to reserve

- 4 cores (8 logical CPUs) for the host IO controllers,
- either 4 (for 1 VM) or 8 (for 2 VMs) cores or threads for the IO of the VMs,
- and 1 core for system management

That still leaves $128-8-4-2=114$ (for 1 VM) or $128-8-8-2=100$ CPUs available. On the other hand, if the goal is to run many small VMs, “losing”, for example, one CPU per VM (plus, again, 8 for IO controllers and 2 for management) means not being able to start more than $128-8-2=118/2=59$ VMs.

To summarize, considering the focus of this paper on CPU and memory intensive workloads, the recommendation is to exclusively assign to the host OS **one** physical CPU per NUMA node. Referring to the hardware shown in *Figure 1, "EPYC Topology"*, that means 8 physical cores (equivalent to 16 logical CPUs) should be assigned.

Allocating Resources to the host OS on KVM

When using KVM, sparing 8 cores and 12 GB of RAM for the host OS is done by stopping the creation of VMs when the total number of vCPUs for these VMs has reached 120, and when the total cumulative amount of allocated RAM has reached 244 GB.

Following all the advice and recommendations from this paper (including the ones given later about VM configuration) will automatically make sure that the unused CPUs are available to the host OS. There are many other ways to enforce this (for example with `cgroups` and `cpusets`). These methods are not described in this paper.

Allocating Resources to the host OS on Xen

When using Xen, host OS (also called “Domain 0” or “Dom0”) resource allocation needs to be done explicitly, at system boot time. Giving 8 physical cores and 12 GB of RAM to Dom0 is done by specifying the following additional parameters on the hypervisor boot command line (for example, by properly editing `/etc/defaults/grub`, and then updating the boot loader):

```
dom0_mem=12288M,max:12288M dom0_max_vcpus=16
```

The number 16 comes from the reservation of 8 physical cores, which, because of hyperthreading, are 16 logical CPUs. 12288 memory (= = 12 GB, in megabytes (MB)) is specified twice to prevent Dom0 from using ballooning, which is not recommended (see https://wiki.xenproject.org/wiki/Tuning_Xen_for_Performance#Memory).

Making sure that Dom0 vCPUs run on specific pCPUs is not strictly necessary. It can be enforced, but since Dom0 is a (special) VM this is only possible via the Xen scheduler. There is no mechanism to communicate this configuration at Xen boot time. Consequently it must be done when the system is live, by specifying the vCPU affinity of Dom0’s vCPUs. Using the Xen’s default `xl` toolstack, it looks as follows:

```
xl vcpu-pin 0 0 0
xl vcpu-pin 0 1 1
xl vcpu-pin 0 2 8
xl vcpu-pin 0 3 9
...
xl vcpu-pin 0 12 48
xl vcpu-pin 0 13 49
xl vcpu-pin 0 14 56
xl vcpu-pin 0 15 57
```

Or, using libvirt’s `virsh`, it looks as follows:

```
virsh vcpupin 0 --vcpu 0 --cpulist 0
virsh vcpupin 0 --vcpu 1 --cpulist 1
virsh vcpu-pin 0 --vcpu 2 --cpulist 8
virsh vcpu-pin 0 --vcpu 3 --cpulist 9
...
virsh vcpupin 0 --vcpu 12 --cpulist 48
```

```
virsh vcpupin 0 --vcpu 13 --cpulist 49
virsh vcpupin 0 --vcpu 14 --cpulist 56
virsh vcpupin 0 --vcpu 15 --cpulist 57
```

As mentioned above, this cannot be done via boot time parameters, and must happen after the system is booted. However, it can be automated via a custom init script (`virsh vcpupin --config ...` is not effective for Dom0).

If you want to limit Dom0 to only a specific (set of) NUMA node(s), the `dom0_nodes=<nodeid>` boot command line option can be used. This will affect both memory and vCPUs. This means that memory of Dom0 will be allocated on the specified node(s), and the vCPUs of Dom0 will be restricted to run on those same node(s). It is possible to change on-line on what pCPUs you want Dom0's vCPUs to run (as shown either via `xl vcpu-pin` or `virsh vcpupin`), but its memory will always stay where it was allocated during boot. On EPYC, at least for the purposes and the scope of this paper, this option is not recommended.

11.4.2 (Transparent) Huge Pages

For virtualization workloads, rather than using Transparent Huge Pages on the host, it is recommended that huge pages (1 GB size, if possible) are used for the memory of the VMs. This sensibly reduces the overhead and the resource contention occurring when a VM updates its own page tables. It is extremely unlikely that the host OS runs a workload which requires or benefits from using (Transparent) Huge Pages. Having them on the host may even negatively affect performance if the THP daemon interferes with the VMs' execution, consuming CPU time and causing latencies. Therefore, it is advised to disable THP on KVM, by adding the following host kernel command-line option:

```
transparent_hugepage=never
```

Another option is executing the following at runtime:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

For being able to use 1 GB Huge Pages as backing memory of KVM guests, such pages need to be allocated on the host, by the host OS. It is best to do that at boot time, as follows:

```
default_hugepagesz=1GB hugepagesz=1GB hugepages=<number of hugepages>
```

The value for `<number of hugepages>` can be computed by taking the amount of memory devoted to VMs, and dividing it by the page size (1 GB). For example, the host in the example scenario has 256 GB RAM; take out 5 percent \sim 26 GB, and the number you get is 230 x 1 GB Huge Pages.

On Xen, none of the above actions is necessary. Dom0 is a paravirtualized guest, for which Huge Pages support is not present. On the other hand, memory used by the hypervisor, and memory allocated for HVM VMs, uses Huge Pages as much as possible by default, so no explicit tuning is needed.

11.4.3 Automatic NUMA Balancing

On Xen, *Automatic NUMA Balancing (NUMAB)* for the host OS should be disabled. Dom0 is a paravirtualized guest without a (virtual) NUMA topology, thus it would be totally useless. Since the Dom0 OS does not detect any NUMA topology, NUMAB will stay off, without any intervention needed.

On KVM, NUMAB can be useful and improve throughput. For example, this can be the case in dynamic virtualization scenarios, where VMs are created, destroyed and re-created relatively quickly, and without statically partitioning and pre-assigning resources (pCPUs and memory) to them. However, latency is introduced, and NUMAB operation can interfere, and cause performance degradation with VMs not needing and not using its services. Furthermore, since this paper focuses on careful and tailored resource pre-allocation, it is recommended to switch NUMAB off. This can be done by adding the following parameter to the host kernel command line:

```
numa_balancing=disable
```

If anything changes and the system is repurposed to achieve different goals, NUMAB can be enabled on-line with the command:

```
echo 0 > /proc/sys/kernel/numa_balancing
```

11.4.4 Services, Daemons and Power Management

The service daemons that have been discussed already in the first part of the paper also run on the host OS of a virtualization system. For them, the same considerations that were made there apply here.

For example, `tuned` should either not be used, or the profile should be set to one that does not implicitly put the CPUs in polling mode. Both throughput-performance and virtualization-host profiles from SUSE Linux Enterprise Server 12 SP3 are ok, from this point of view, as neither of them touches `/dev/cpu_dma_latency`. `irqbalance` can be a source of latency, for no significant performance improvement. Thus the suggestion is again to disable it (but then, IRQs may need to be manually bound to the appropriate CPUs, considering the IO topology).

As far as power management is concerned, the `cpufreq` governor can either be kept as it is by default, or switched to `performance`, following the previous advice, based on the nature of the workloads of interest.



Note: Power Management

For anything that concerns power management, on KVM, changing the tuned profile, or using `cpupower`, from the host OS will have the same effect described in the first part of the paper. On Xen, however, CPU frequency scaling is enacted by the hypervisor. It can be controlled from within Dom0, by using a different tool, called `xenpm`, like in the example below:

```
xenpm set-scaling-governor performance
```

11.5 Resource Allocation and Tuning of the VMs

For instructions of how to create an initial VM configuration, run the VM, and install a guest OS, refer to the SUSE documentation at https://www.suse.com/documentation/sles-12/book_virt/data/cha_kvm_inst.html .

From a VM configuration perspective, the two most important factors for achieving top performance on CPU and memory bound workloads are:

1. Placement of the VM on top of the host resources
2. Enlightenment of the VM about its own topology

The former factor is critical. For example, with two VMs, one should be run on each socket to maximize CPU and memory access parallelism. The latter also helps, in particular with big VMs, which span more than one of the EPYC NUMA nodes. When the VM is made aware of its own virtual NUMA topology, all the tuning actions described in the first part of this paper become applicable to the workloads the VM is running.



Note: Additional Tuning Factors

Even with these two factors being the most important aspects of the VM configuration tuning process, there are additional factors which should be considered.

11.5.1 Placement of VMs

When a VM is created, memory is allocated on the host to act as its virtual RAM. This is generally something that happens at VM boot time. This either cannot be changed at all, or cannot be changed without a price. Therefore, it is of paramount importance to get this initial placement correct. Both Xen and KVM can make “educated guesses” on what a good placement might be. However, this paper provides advise only on how to **manually** achieve the best possible placement, considering EPYC specific characteristics.

Where the vCPUs will run (this means, on what pCPUs) is also decided at VM creation time. Contrarily to memory, it is less of a problem to change this when the VM is running, but it is still better to start the VM directly with good vCPU placement. This is particularly true on Xen, where vCPU placement actually drives and controls memory placement.

Since this paper does not consider oversubscribed scenarios, this form of static resource assignment is particularly effective. However, similar principles apply even when oversubscription is present.

Placement of memory happens by means of the `<numatune>` XML element:

```
<numatune>
  <memory mode='strict' nodeset='0-7' />
  <memnode cellid='0' mode='strict' nodeset='0' />
  <memnode cellid='1' mode='strict' nodeset='1' />
  ...
</numatune>
```

The parameter `'strict'` enforces the memory to be allocated where it is specified. A cell is a virtual NUMA node, with `cellid` being its ID, and `nodeset` telling on what host physical NUMA node its memory must be put. For NUMA-unaware VMs, this can still be used, but it will have only one `<memnode>` element.

Placement of vCPUs happens via the `<cputune>` element, as in the example below:

```
<vcpu placement='static'>96</vcpu>
<cputune>
  <vcupin vcpu='0' cpuset='1' />
  <vcupin vcpu='1' cpuset='65' />
  <vcupin vcpu='2' cpuset='2' />
  <vcupin vcpu='3' cpuset='66' />
  <vcupin vcpu='4' cpuset='3' />
  ...
</cputune>
```

In this example, in the `<vcpupin>` elements, `vcpu` is the vCPU ID, and `cpuset` defines on what host physical CPU it should be bound to.

When “pinning” vCPUs to pCPUs, it is generally wise to put adjacent vCPU IDs (like vCPU 0 and vCPU 1) on actual host hyperthread siblings (like pCPU 1 and pCPU 65) on the test server. QEMU uses a static hyperthread sibling CPU ID assignment. Thus, by doing as described, at the end the virtual hyperthread siblings will run on real hardware hyperthread siblings.

The following paragraphs will address several scenarios with varying number and sizes of VMs. All the examples detailed in this section refer to the topology shown in *Figure 1, “EPYC Topology”*.

One Very Large VM

It is possible to use “just one” VM on the EPYC server. Reasons for this scenario include security/isolation, flexibility, high availability, and others. In these cases typically a single large VM would be used, almost as large as the host itself. Consider a VM with 96 vCPUs and 200 GB of RAM. That is a VM that spans multiple host NUMA nodes. It is recommended to create eight virtual NUMA nodes, that is as many as there are physical NUMA nodes, and divide the VM’s memory equally among them. The 96 vCPUs can be divided into 12 assigned to each node. It is also recommended to use full cores, this means: assign vCPUs 0 and 1 to Core P#1 in *Figure 1, “EPYC Topology”*, vCPUs 2 and 3 to Core P#2, vCPUs 4 and 5 to Core P#5. This configuration pins vCPUs 0 and 1 to pCPUs 0 and 64, vCPUs 2 and 3 to pCPUs 1 and 65, etc.

Memory should be split equally among all 8 nodes, and a virtual topology will be provided to the guest OS of the VM. Using this setup, each of the VM’s vCPUs will access its own memory directly, and use InfinityFabric links to reach foreign memory, as it happens on the host. Workloads inside such a VM can be tuned exactly like they were running on a bare metal EPYC server (however on one with slightly fewer CPUs and less RAM).

The following example `numactl` output comes from a VM configured as explained:

```
available: 8 nodes (0-7)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11
node 0 size: 25118 MB
node 0 free: 25000 MB
node 1 cpus: 12 13 14 15 16 17 18 19 20 21 22 23
node 1 size: 25198 MB
node 1 free: 25116 MB
node 2 cpus: 24 25 26 27 28 29 30 31 32 33 34 35
node 2 size: 25198 MB
node 2 free: 25122 MB
node 3 cpus: 36 37 38 39 40 41 42 43 44 45 46 47
```

```

node 3 size: 25198 MB
node 3 free: 25106 MB
node 4 cpus: 48 49 50 51 52 53 54 55 56 57 58 59
node 4 size: 25198 MB
node 4 free: 25109 MB
node 5 cpus: 60 61 62 63 64 65 66 67 68 69 70 71
node 5 size: 25198 MB
node 5 free: 25122 MB
node 6 cpus: 72 73 74 75 76 77 78 79 80 81 82 83
node 6 size: 25198 MB
node 6 free: 25116 MB
node 7 cpus: 84 85 86 87 88 89 90 91 92 93 94 95
node 7 size: 25196 MB
node 7 free: 25111 MB
node distances:
node  0  1  2  3  4  5  6  7
  0: 10 20 20 20 20 20 20 20
  1: 20 10 20 20 20 20 20 20
  2: 20 20 10 20 20 20 20 20
  3: 20 20 20 10 20 20 20 20
  4: 20 20 20 20 10 20 20 20
  5: 20 20 20 20 20 10 20 20
  6: 20 20 20 20 20 20 10 20
  7: 20 20 20 20 20 20 20 10

```

This is analogous to the host topology already presented in the paper, with the only differences being the number and the IDs of the CPUs, and the nodes' distance table. Unfortunately, the `libvirt` version available in SUSE Linux Enterprise Server 12 SP3 does not allow to define virtual node distances (while later versions do).

See [Section 15, "Appendix A"](#) for an (almost) complete VM configuration file.

Two Large VMs

When running two VMs with 48 vCPUs and 100 GB memory each, nearly the same configuration can be used, but each VM should be placed on one of the EPYC sockets. This means that each VM will span – both vCPU- and memory-wise – 4 NUMA nodes (and hence have 4 virtual NUMA nodes). The reason behind locating one on each socket is that workloads running within each VM will not need to use the inter-socket interconnect, to access memory.

In this example scenario, the `numactl` output looks as follows:

```

available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11

```

```

node 0 size: 25118 MB
node 0 free: 25026 MB
node 1 cpus: 12 13 14 15 16 17 18 19 20 21 22 23
node 1 size: 25198 MB
node 1 free: 25094 MB
node 2 cpus: 24 25 26 27 28 29 30 31 32 33 34 35
node 2 size: 25198 MB
node 2 free: 25084 MB
node 3 cpus: 36 37 38 39 40 41 42 43 44 45 46 47
node 3 size: 25196 MB
node 3 free: 25108 MB
node distances:
node  0  1  2  3
  0: 10 20 20 20
  1: 20 10 20 20
  2: 20 20 10 20
  3: 20 20 20 10

```

Four to Eight Medium-size VMs

The same principle adopted for two VMs is followed in a scenario with four VMs, with 24 vCPUs and 50 GB memory. In this case, VM1 should be put on nodes 0 and 1, VM2 on nodes 3 and 4, etc. Single VMs again span multiple (two, in this case) host NUMA nodes, but do not cross the socket boundary. Since they span two nodes, they benefit from being NUMA-aware.

In a scenario with eight VMs, with 12 vCPUs and 25 GB RAM each, each VM can be put on a single host NUMA node. In this case, therefore, there is no need for the VMs to be NUMA-aware. It is still helpful to let the guest OS know about the guest specific characteristics of the pCPU being used (cores, threads, etc), but it is less performance-critical (at least for memory intensive workloads).

The example below shows the `numactl` output from one of these eight NUMA-unaware VMs:

```

available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11
node 0 size: 25117 MB
node 0 free: 24888 MB
node distances:
node  0
  0: 10

```

Many “Micro”-VMs

When there is the need to have more than one VM per host NUMA node, the VMs will also not be NUMA-aware. If they have 4 vCPUs each, the best solution is to assign VM1 to Core P#0 and Core P#1, and VM2 to Core P#2 and Core P#3, on NUMA node 1, and do the same on the other nodes. That means VMs will use hyperthreading and, if possible, they should be made aware of this bit of topology information.

If they must have two vCPUs each, but they are still not more than 128 VMs, it is best to assign VM1 to Core P#0-PU P#0 and Core P1-PU P#2, then VM2 to Core P#2-PU P#4 and Core P3-PU P#6, on node 1 (and so on for the other nodes). This means only one of the hyperthread siblings on each core is used. The other sibling can be left idle, or be used for the IO of the VMs themselves (by giving them to the host OS, and using them for running either the emulator’s IO threads, on KVM, or the IO back-ends, on Xen).

When using 128 VMs, the recommendation is to use single cores for each VMs (although, this time, VM1 should be assigned to Core P#0, VM2 to Core P#2, etc.).

It should be avoided to have vCPUs from different VMs running on two hyperthread siblings. This is supported and works, but is unideal for performance, especially from a consistency point of view, as the “speed” of the vCPUs of one VM will depend on what the vCPUs of another VM are doing. It is also a less secure setup, as running on hyperthread siblings may make it easier to enact cross-VM side channel attacks.

As far as memory is concerned, it is recommended that the memory of the VMs that are assigned to a NUMA node resides on that same node.

Oversubscription

If an oversubscription scenario is wanted, exclusive 1-to-1 vCPU to pCPU assignment may be unideal. In this case, it is in generally better to let the hypervisor scheduler take advantage of any idle interval on a wide range of pCPUs, and use that to execute as many vCPUs as possible for as long as it can. However, leaving all the vCPUs of all the VMs completely free to run on any pCPU might be equally bad, especially on EPYC. That may quickly put the system in a state where a lot of VMs mostly access memory from remote NUMA nodes, with regard to where their vCPUs are running.

In this case the recommendation is to try to partition the overall workload. This can be done by assigning groups of VMs to single nodes, or to the smallest possible set of nodes, in a way that takes load into account. This happens for example by avoiding to put all the CPU intensive VMs on the same node, or by avoiding to overload a node and leaving others lightly loaded or idle, etc.

This grouping of VMs on (a set of) nodes can still be done with vCPUs affinity. But there are also other mechanisms, specifically designed for doing “pooling”, such as `cgroups` (on KVM) and `cpupools` (on Xen). On Xen, there is also a feature called *soft vCPU affinity*, which can be used together with “traditional” vCPU affinity (also called *hard vCPU affinity*) as a finer grained and more powerful way of controlling resource allocation in such a scenario.

11.5.2 Enlightenment of the VMs

“Enlightment” means letting the guest OS know as many details as possible of the (virtual) topology of the VM. Of course, this brings performance improvements only if such topology is properly and effectively mapped on host resources, and if the mapping is stable.

To ensure the VM has a vCPU topology, use the following:

```
<cpu mode='host-passthrough'>
  <topology sockets='2' cores='24' threads='2' />
  <numa>
    <cell id='0' cpus='0-11' memory='26214400' unit='KiB' />
    <cell id='1' cpus='12-23' memory='26214400' unit='KiB' />
    <cell id='2' cpus='24-35' memory='26214400' unit='KiB' />
    ...
  </numa>
</cpu>
```

The `<topology>` element specifies the CPU characteristics, while each `<cell>` element defines one virtual NUMA node.

The following example (available on KVM only) is also useful, especially for VMs that span multiple host NUMA nodes, but in general every time that vCPUs are pinned to pCPUs that share a cache layer:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  ...
  <qemu:commandline>
    <qemu:arg value='-cpu' />
    <qemu:arg value='host,migratable=off,+invtsc,l3-cache=on' />
  </qemu:commandline>
</domain>
```



```
</qemu:commandline>
</domain>
```

The element `l3-cache=on` may significantly reduce resource contention within the VM, when the guest OS scheduler wants to wake up a task (while `migratable=off` is necessary for QEMU to preserve the other passed flags).

11.5.3 Memory Backing

The VMs need to be told to use the huge pages that were reserved for them. To effectively use 1 GB huge pages, the amount of memory each VM is given should be a multiple of 1 GB. Also, *Kernel Same Page Merging (KSM)* should be disabled. This is done as follows:

```
<memory unit='KiB'><memory in KB></memory>
  <memoryBacking>
    <hugepages>
      <page size='1048576' unit='KiB' />
    </hugepages>
    <nosharepages />
  </memoryBacking>
```



Note: Huge Pages on Xen

On Xen, for HVM guests, huge pages are used by default, so the `<memoryBacking>` element is technically not necessary.

11.5.4 No Ballooning

To get the full benefit of using huge pages, memory ballooning must be disabled. If the ballooning driver is not huge-pages-aware, using it would split the pages and fragment memory. Disable memory ballooning as follows :

```
<currentMemory unit='KiB'><memory in KiB></currentMemory>
```

Specify the same amount of memory as in the `<memory>` element and never change the memory of the VM at runtime.

11.5.5 (Transparent) Huge Pages

If huge pages are used for allocating the VMs' memory on the host, they can also be used inside the VMs, either explicitly, or via THP. Whether that helps performance is workload dependant. The analysis and the considerations made in the first part of the paper about using (T)HP on bare metal can also be applied here.

11.5.6 Automatic NUMA Balancing

Similarly to THP, if the VM is NUMA-aware, NUMAB can be used inside of it to boost the performance of NUMA-unaware workloads.

11.5.7 Services and Daemons

The `irqbalance` tool can be a source of latency inside of the VM, because of the way it uses the `/proc/interrupts` interface. For workloads that are particularly sensitive to latency, consider disabling it within the VMs (of course by taking the appropriate alternative measures, like binding IRQs, if necessary).

11.5.8 Emulator IO Threads / Disaggregation

IO for the VMs is carried out either by emulators, or by the so-called *back-ends* of paravirtualized drivers. Both the IO threads of the emulators, and the back-ends are user or kernel threads running in the host OS. As such, they can run on specific subsets of the host OS' CPUs (Dom0's virtual CPUs', in the case of Xen). For example, a one vCPU VM can have its vCPU bound to a hyperthread sibling, while the IO threads can be bound to the other sibling. Considering that a common execution pattern will be for the vCPU to be idle, when the IO threads are busy, this setup maximizes the exploitations of hardware resources.

On Xen, there is also the possibility of setting up driver domains. These are special guests which act as back-ends of a particular IO device for one or more VMs. In case they are used, make sure that such guests run close enough to both the hardware they are providing their abstraction for, and the VMs that are servicing.

11.6 Test Workload: STREAM

To show the validity of some of the tuning advise given, the STREAM benchmark is used again.

Test Scenario: One Large VM

Figure 4 shows the bandwidth achieved by STREAM, using a single thread, on the host and inside one large VM, in the following configurations:

- No topology: the VM is not provided any virtual NUMA or CPU topology, nor are its memory and vCPUs assigned to host nodes and pCPUs
- Topology: the VM is provided the virtual NUMA and CPU topology described in the above section, but still no mapping and pinning of memory and vCPUs
- Topology Tuned: the VM is provided its topology, and memory is allocated and vCPUs are pinned as recommended in the tuning section

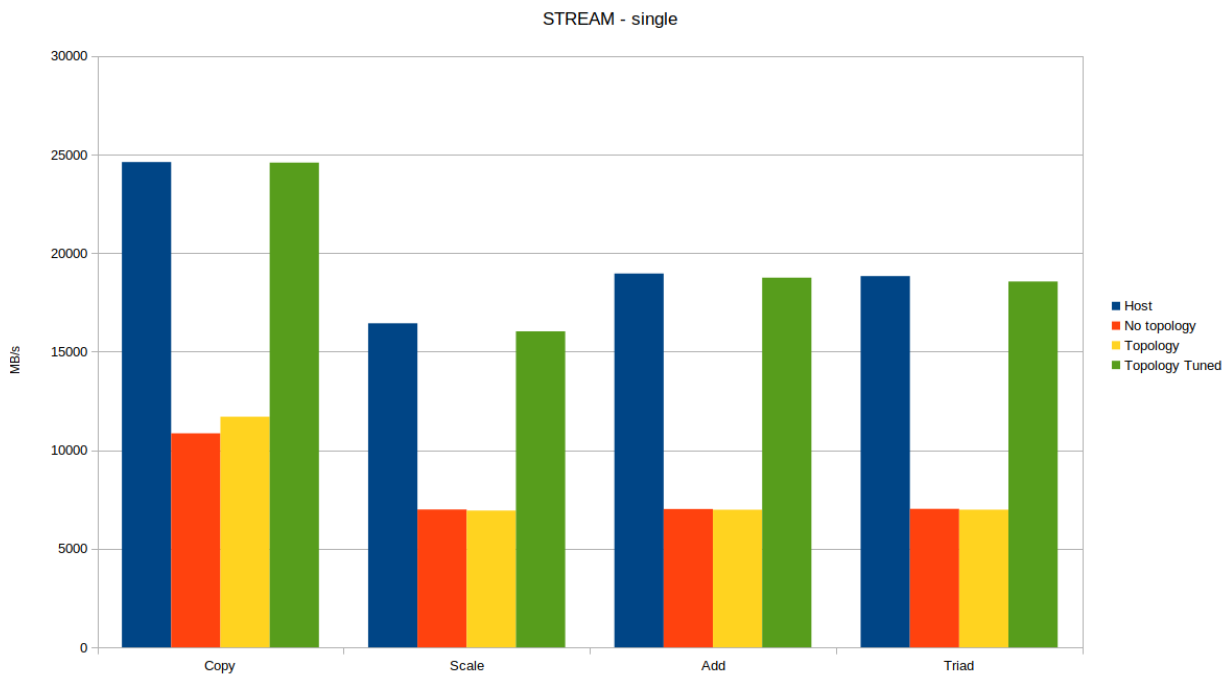


FIGURE 4: STREAM BANDWIDTH - SINGLE THREAD IN ONE VM

It appears evident how resource allocation is critical, for achieving good performance inside of the VM. When mapping VM resources on host resources as recommended, almost the same results are reached from within the VM, as obtained on the host.

Figure 5 shows the same setup, but when 16 threads are used.

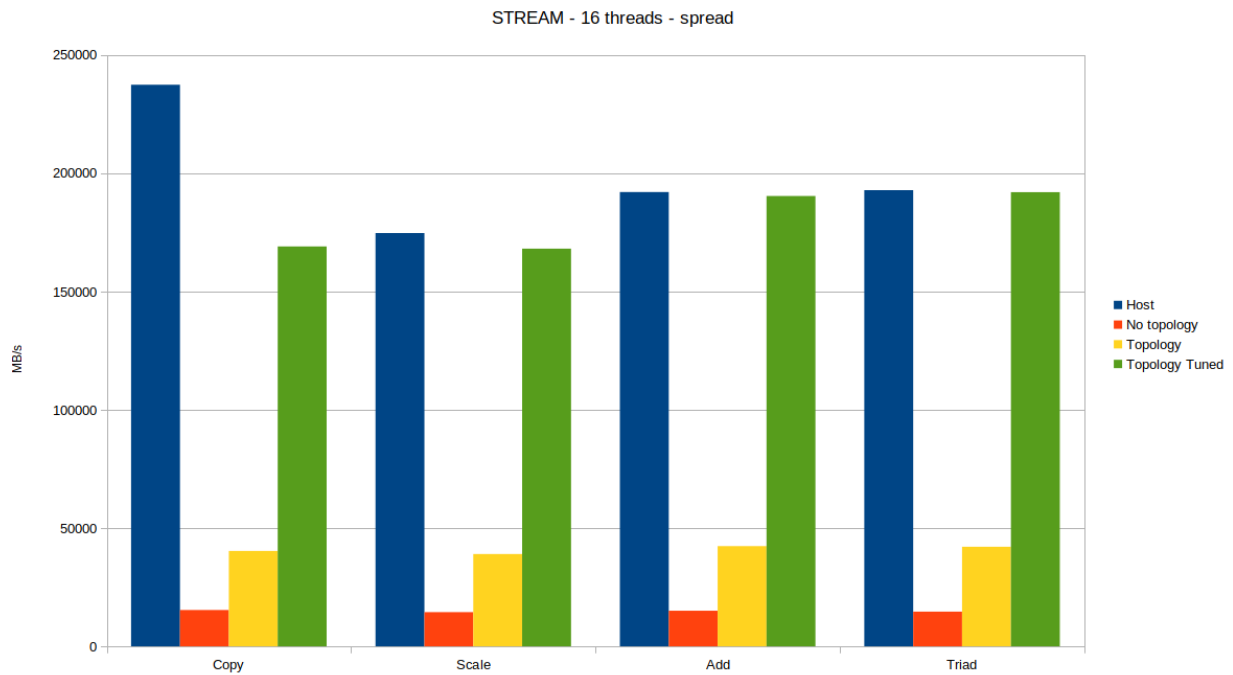


FIGURE 5: STREAM BANDWIDTH - 16 THREADS IN ONE VM

In this case, providing the VM with a meaningful topology already improves the performance. However, the result is still far from ideal, and good results are reached only when doing both topology enlightenment and proper placement of the VM.



Note: “Copy” in Figure 5

With full tuning applied, we expected results matching the ones on the host. That was the case for “Scale”, “Add” and “Triad”. “Copy”, however, is implemented using glibc's `memcpy()` which, when running on the host, is optimized with non-temporal store and prefetch instructions. In the VM, this does not happen because the heuristics that enables the use of those instructions does not trigger. This happens because the VM does not have any information available about how many and which logical CPUs share the L3 caches (which is what drives the heuristics). At the time of writing this paper, there is no way to let the VM have this information. SUSE is internally tracking this issue (partners can look up bug #1091081). When this is resolved, the performance of the “Copy” operation in a VM should reach the level of its performance on the host.

Test Scenario: Two Large VMs

Figure 6 shows how effective it is, when using two large VMs, to place them on separate sockets, as recommended above:

- VM1 alone: is the throughput achieved when running **only one** of the two VMs
- Both, VM1: is the throughput achieved on VM1, when running both the VMs concurrently
- Both, VM2: is the throughput achieved on VM2, when running both the VMs concurrently
- Both, VM1 + VM2: is the aggregate throughput, that is, the sum of the throughput achieved on VM1 and on VM2 (when running both of them concurrently)

In this case, the VMs have 48 vCPUs (each). Note how, based on the recommended tuning:

- performance achieved in both the VMs, when they are running concurrently, is the same as when only one of them is running alone;
- the aggregate throughput is, on Triad, 194 GB/s; on the host and on only one large VM, it was 192 GB/s.

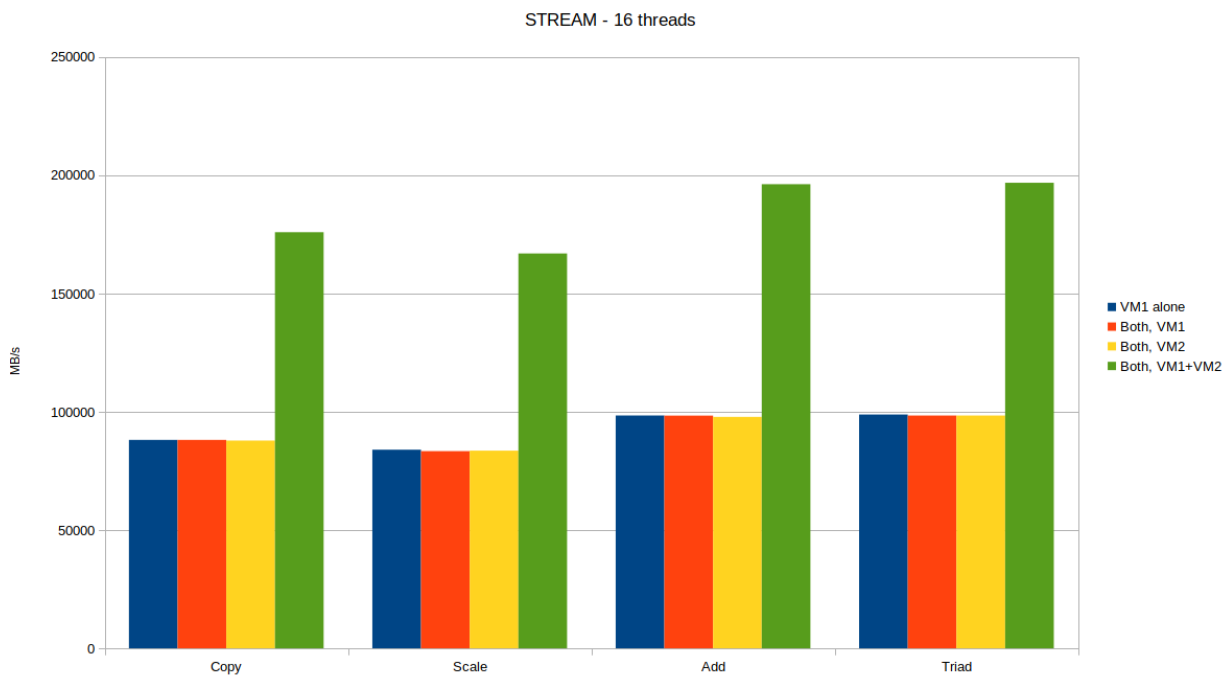


FIGURE 6: STREAM BANDWIDTH - 16 THREADS IN TWO VMs

Test Scenario: Four to Eight Medium-Size VMs

Finally, Figure 7 shows again the effectiveness of carefully tuned resource allocation, this time on NUMA-unaware VMs. For this experiments, two VMs are used, each with 12 vCPUs and 25 GB memory. There are only two VMs used for simplicity, but the test hardware used could accommodate eight of them (without violating the recommendation of leaving at least one core on each node for the host OS). In this case, there is no virtual NUMA topology to construct for the VMs, and only 8 threads are used:

- VM1, unpinned: is the throughput achieved on VM1 running alone, when the VM's vCPUs and memory are not pinned to the host resources
- VM1, pinned: is the throughput achieved on VM1 running alone, when the VM's vCPUs and memory are pinned to the host resources
- VM1 + VM2, unpinned: is the aggregate throughput reached together by VM1 and VM2, both not pinned in any way to the host
- VM1 + VM2, pinned: is the aggregate throughput reached together by VM1 and VM2, when both are pinned to the host resources

In the “pinned” cases, the VMs are assigned to one NUMA node each, from different sockets. It is again evident how much reasonable placement of VM resources helps performance, even in case of NUMA-unaware VMs.

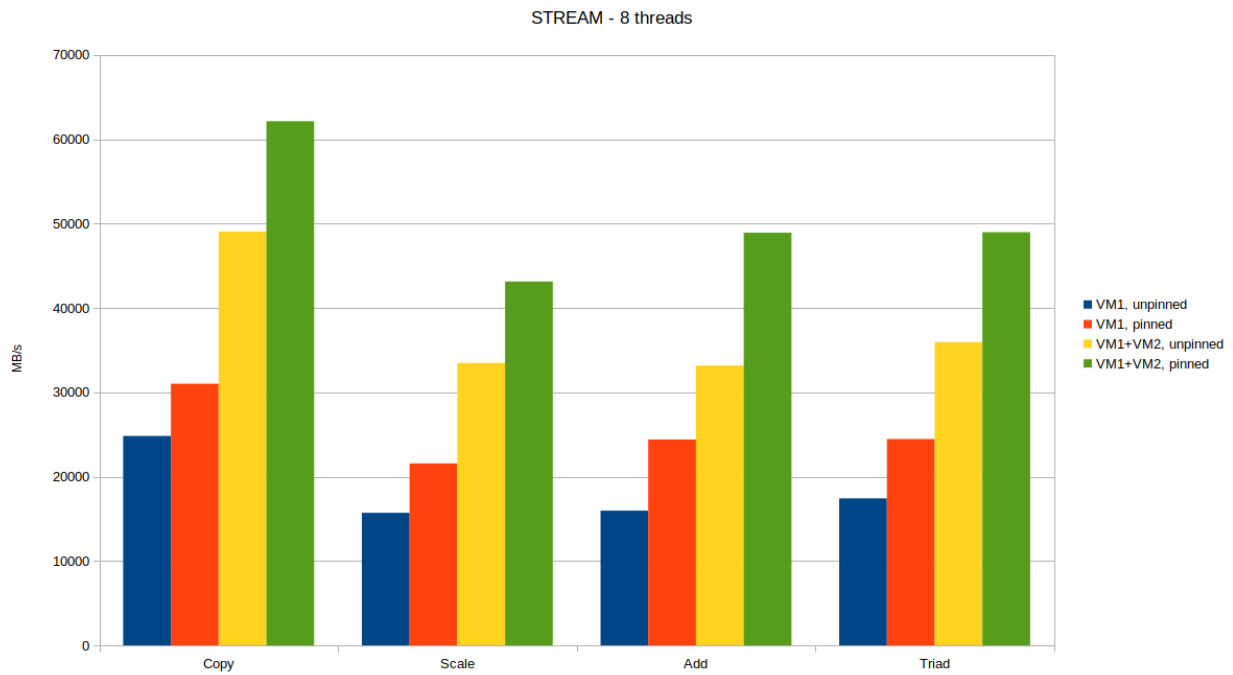


FIGURE 7: STREAM BANDWIDTH - 8 THREADS IN TWO VMS

12 Conclusion

The introduction of EPYC pushes the boundaries of what is possible for memory and IO-bound workloads with much higher bandwidth and available number of channels. A properly configured and tuned workload can exceed the performance of many contemporary off-the-shelf solutions even when fully customized. The symmetric and balanced nature of the machine makes the task of tuning a workload considerably easier given that each partition can have symmetric performance.

With SUSE Linux Enterprise, all the tools to monitor and tune a workload are readily available. Your customers can extract the maximum performance and reliability running their applications, either on bare metal or virtualized, on the EPYC platform.

13 Resources

For more information, refer to:

- AMD SenseMI Technology (<https://www.amd.com/en/technologies/sense-mi>)
- Balanced power plan optimized for AMD Ryzen processors (<https://community.amd.com/community/gaming/blog/2017/04/06/amd-ryzen-community-update-3>)
- EPYC Tech Day: Gerry Talbot (<https://www.youtube.com/watch?v=W5lhEit6NqY>)
- Optimizing Linux for Dual-Core AMD Opteron Processors (<http://www.novell.com/traininglocator/partners/amd/4622016.pdf>)
- Systems Performance: Enterprise and the Cloud by Brendan Gregg (<http://www.brendan-gregg.com/sysperfbook.html>)
- NASA Parallel Benchmark (<https://www.nas.nasa.gov/publications/npb.html>)

14 Glossary

CPU-bound	An application whose primary bottleneck is the computation power at maximum speed of a CPU. The simple case is a single-thread application that keeps the CPU at 100 percent utilization. A more complex example is a multi-threaded application where N threads keep N CPUs at 100 percent utilization.
C-State	The idle state of the CPU where lower states use less power but have larger exit latencies.
HPC	High Performance Computing
Memory-bound	An application whose performance is limited by the maximum bandwidth of memory. It may be an application whose working set size exceeds the size of the L3 cache, in the case of EPYC, or an application whose working set size exceeds the size of a NUMA node.
MCM	Multi-Chip Module
NUMA	Non-Uniform Memory Architecture

NUMA node	A mapping between a set of CPUs and a range of memory where the cost of access to main memory is a fixed value, and a direct link, that is considered “local”.
PSS	Proportional Set Size is the estimated amount of physical memory used in a mapping when sharing between processes is taken into account.
P-State	Performance State is a selected frequency and voltage a CPU is running with.
RSS	Resident Set Size is the amount of physical memory used in a mapping.
SMP	Symmetric multiprocessing
THP	Transparent Huge Pages
WSS	Working Set Size is the estimated amount of memory a process needs within a period to operate without paging.
VSZ	Virtual Size of a memory mappings

15 Appendix A

Example of a VM configuration file:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
<name>sles12sp3_01</name>
  <uuid>26137bb8-9e5f-48e9-a81d-63ae36400196</uuid>
  <memory unit='KiB'>209715200</memory>
  <currentMemory unit='KiB'>209715200</currentMemory>
  <memoryBacking>
    <hugepages>
      <page size='1048576' unit='KiB' />
    </hugepages>
    <nosharepages />
  </memoryBacking>
  <vcpu placement='static'>96</vcpu>
  <cputune>
    <vcupin vcpu='0' cpuset='1' />
```

```
<vcpupin vcpu='1' cpuset='65' />
<vcpupin vcpu='2' cpuset='2' />
<vcpupin vcpu='3' cpuset='66' />
<vcpupin vcpu='4' cpuset='3' />
<vcpupin vcpu='5' cpuset='67' />
<vcpupin vcpu='6' cpuset='4' />
<vcpupin vcpu='7' cpuset='68' />
<vcpupin vcpu='8' cpuset='5' />
<vcpupin vcpu='9' cpuset='69' />
<vcpupin vcpu='10' cpuset='6' />
<vcpupin vcpu='11' cpuset='70' />
<vcpupin vcpu='12' cpuset='9' />
<vcpupin vcpu='13' cpuset='73' />
<vcpupin vcpu='14' cpuset='10' />
<vcpupin vcpu='15' cpuset='74' />
<vcpupin vcpu='16' cpuset='11' />
<vcpupin vcpu='17' cpuset='75' />
<vcpupin vcpu='18' cpuset='12' />
<vcpupin vcpu='19' cpuset='76' />
<vcpupin vcpu='20' cpuset='13' />
<vcpupin vcpu='21' cpuset='77' />
<vcpupin vcpu='22' cpuset='14' />
<vcpupin vcpu='23' cpuset='78' />
<vcpupin vcpu='24' cpuset='17' />
<vcpupin vcpu='25' cpuset='81' />
<vcpupin vcpu='26' cpuset='18' />
<vcpupin vcpu='27' cpuset='82' />
<vcpupin vcpu='28' cpuset='19' />
<vcpupin vcpu='29' cpuset='83' />
<vcpupin vcpu='30' cpuset='20' />
<vcpupin vcpu='31' cpuset='84' />
<vcpupin vcpu='32' cpuset='21' />
<vcpupin vcpu='33' cpuset='85' />
<vcpupin vcpu='34' cpuset='22' />
<vcpupin vcpu='35' cpuset='86' />
<vcpupin vcpu='36' cpuset='25' />
<vcpupin vcpu='37' cpuset='89' />
<vcpupin vcpu='38' cpuset='26' />
<vcpupin vcpu='39' cpuset='90' />
<vcpupin vcpu='40' cpuset='27' />
<vcpupin vcpu='41' cpuset='91' />
<vcpupin vcpu='42' cpuset='28' />
<vcpupin vcpu='43' cpuset='92' />
<vcpupin vcpu='44' cpuset='29' />
<vcpupin vcpu='45' cpuset='93' />
<vcpupin vcpu='46' cpuset='30' />
<vcpupin vcpu='47' cpuset='94' />
```

```
<vcpupin vcpu='48' cpuset='33' />
<vcpupin vcpu='49' cpuset='97' />
<vcpupin vcpu='50' cpuset='34' />
<vcpupin vcpu='51' cpuset='98' />
<vcpupin vcpu='52' cpuset='35' />
<vcpupin vcpu='53' cpuset='99' />
<vcpupin vcpu='54' cpuset='36' />
<vcpupin vcpu='55' cpuset='100' />
<vcpupin vcpu='56' cpuset='37' />
<vcpupin vcpu='57' cpuset='101' />
<vcpupin vcpu='58' cpuset='38' />
<vcpupin vcpu='59' cpuset='102' />
<vcpupin vcpu='60' cpuset='41' />
<vcpupin vcpu='61' cpuset='105' />
<vcpupin vcpu='62' cpuset='42' />
<vcpupin vcpu='63' cpuset='106' />
<vcpupin vcpu='64' cpuset='43' />
<vcpupin vcpu='65' cpuset='107' />
<vcpupin vcpu='66' cpuset='44' />
<vcpupin vcpu='67' cpuset='108' />
<vcpupin vcpu='68' cpuset='45' />
<vcpupin vcpu='69' cpuset='109' />
<vcpupin vcpu='70' cpuset='46' />
<vcpupin vcpu='71' cpuset='110' />
<vcpupin vcpu='72' cpuset='49' />
<vcpupin vcpu='73' cpuset='113' />
<vcpupin vcpu='74' cpuset='50' />
<vcpupin vcpu='75' cpuset='114' />
<vcpupin vcpu='76' cpuset='51' />
<vcpupin vcpu='77' cpuset='115' />
<vcpupin vcpu='78' cpuset='52' />
<vcpupin vcpu='79' cpuset='116' />
<vcpupin vcpu='80' cpuset='53' />
<vcpupin vcpu='81' cpuset='117' />
<vcpupin vcpu='82' cpuset='54' />
<vcpupin vcpu='83' cpuset='118' />
<vcpupin vcpu='84' cpuset='57' />
<vcpupin vcpu='85' cpuset='121' />
<vcpupin vcpu='86' cpuset='58' />
<vcpupin vcpu='87' cpuset='122' />
<vcpupin vcpu='88' cpuset='59' />
<vcpupin vcpu='89' cpuset='123' />
<vcpupin vcpu='90' cpuset='60' />
<vcpupin vcpu='91' cpuset='124' />
<vcpupin vcpu='92' cpuset='61' />
<vcpupin vcpu='93' cpuset='125' />
<vcpupin vcpu='94' cpuset='62' />
```

```

    <vcpupin vcpu='95' cpuset='126' />
</cputune>
<numatune>
  <memory mode='strict' nodeset='0-7' />
  <memnode cellid='0' mode='strict' nodeset='0' />
  <memnode cellid='1' mode='strict' nodeset='1' />
  <memnode cellid='2' mode='strict' nodeset='2' />
  <memnode cellid='3' mode='strict' nodeset='3' />
  <memnode cellid='4' mode='strict' nodeset='4' />
  <memnode cellid='5' mode='strict' nodeset='5' />
  <memnode cellid='6' mode='strict' nodeset='6' />
  <memnode cellid='7' mode='strict' nodeset='7' />
</numatune>
<os>
  <type arch='x86_64' machine='pc-i440fx-2.9'>hvm</type>
  <boot dev='hd' />
</os>
<features>
  <acpi />
  <apic />
</features>
<cpu mode='host-passthrough' check='none'>
  <topology sockets='8' cores='6' threads='2' />
  <numa>
    <cell id='0' cpus='0-11' memory='26214400' unit='KiB' />
    <cell id='1' cpus='12-23' memory='26214400' unit='KiB' />
    <cell id='2' cpus='24-35' memory='26214400' unit='KiB' />
    <cell id='3' cpus='36-47' memory='26214400' unit='KiB' />
    <cell id='4' cpus='48-59' memory='26214400' unit='KiB' />
    <cell id='5' cpus='60-71' memory='26214400' unit='KiB' />
    <cell id='6' cpus='72-83' memory='26214400' unit='KiB' />
    <cell id='7' cpus='84-95' memory='26214400' unit='KiB' />
  </numa>
</cpu>
...
<devices>
  <emulator>/usr/bin/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' type='qcow2' />
    <source file='/home/sles12sp3_01.img' />
    <target dev='vda' bus='virtio' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' />
  </disk>
  ...
  <interface type='network'>
    <mac address='52:54:00:9e:08:44' />
    <source network='default' />

```

```
<model type='virtio' />
<address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0' />
</interface>
...
<rng model='virtio'>
  <backend model='random'>/dev/urandom</backend>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0' />
</rng>
</devices>
<qemu:commandline>
  <qemu:arg value='-cpu' />
  <qemu:arg value='host,migratable=off,+invtscl3-cache=on' />
</qemu:commandline>
</domain>
```

16 Legal Notice

Copyright ©2006–2018 SUSE LLC and contributors. All rights reserved.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or (at your option) version 1.3; with the Invariant Section being this copyright notice and license. A copy of the license version 1.2 is included in the section entitled “GNU Free Documentation License”.

SUSE, the SUSE logo and YaST are registered trademarks of SUSE LLC in the United States and other countries. For SUSE trademarks, see <http://www.suse.com/company/legal/>. Linux is a registered trademark of Linus Torvalds. All other names or trademarks mentioned in this document may be trademarks or registered trademarks of their respective owners.

This article is part of a series of documents called "SUSE Best Practices". The individual documents in the series were contributed voluntarily by SUSE's employees and by third parties.

All information found in this article has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy.

Therefore, we need to specifically state that neither SUSE LLC, its affiliates, the authors, nor the translators may be held liable for possible errors or the consequences thereof. Below we draw your attention to the license under which the articles are published.

GNU Free Documentation License

Copyright (C) 2000, 2001, 2002 Free Software Foundation, Inc. 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects. If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

```
Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts". line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.