TUNING GUIDE

AMD EPYC™ 9005

OpenShift® Containers Tuning Guide

AMD
together we advance_data center computing

PID: 58724
v1.0
October 2024

Sonemaly Phrasavath and Charles Wong

DATE	VERSION	CHANGES
June, 2024	0.1	Initial NDA release
October, 1.0	1.0	Initial public release

AUDIENCE

This document is intended for a technical audience such as production deployment and performance engineering teams with:

- A background in configuring servers and containers.
- Administrator-level access to both the server management Interface (BMC) and the OS.
- Familiarity with setting up OpenShift clusters.
- Familiarity with both the BMC and OS-specific configuration, monitoring, and troubleshooting tools.

OPENSHIFT® CONTAINERS TUNING GUIDE

CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 1: Introduction

Containerizing an application platform and its associated dependencies abstracts the underlying infrastructure and OS differences for efficiency. Each container is bundled into one package containing an entire runtime environment, including an application with all its dependencies, libraries and other binaries, and configuration files needed to run that application. Containers running applications in a production environment need management to ensure consistent uptime. If a container goes down, then another container needs to start automatically.

OpenShift Container Platform (OCP) enables automated container deployment and management. OCP is a portable and extensible platform for managing containerized workloads and services. It takes care of scaling and failover for your application, provides deployment and more. It has a large, rapidly growing ecosystem.

OpenShift includes the following features:

- Service discovery and load balancing.
- Storage orchestration.
- Automated rollouts and rollbacks.
- Automatic bin packing.
- Self-healing.
- Secret and configuration management.

This tuning guide provides detailed descriptions of OpenShift configuration settings that can optimize containerized application performance on servers powered by AMD EPYC™ 9005 Series processors

1.1 - About Tuning OpenShift

This tuning guide focuses on recommended OpenShift deployment configuration settings that can take advantage of the AMD EPYC CCDs and further enhance performance. This guide supplements the tuning options recommended by Red Hat, which includes optimization guidelines for compute resources, persistent storage, networking, etc. in the Scalability and Performance* chapter. These tuning options are version-specific.

The AMD EPYC processor architecture supports a split L3 Cache topology, which is known externally as a Core Complex Die (CCD). Please see the *AMD EPYC™ 9005 Series Architecture Overview* (available from the AMD Documentation Hub) for detailed information about CCXs, CCDs, and NUMA references.

Additional application-specific BIOS and other settings may further boost performance. See the latest version of the appropriate application-specific tuning guide for additional information.

1.2 - Important Reading

Please be sure to read the following guides (available from the AMD Documentation Hub), which contain important foundational information about 5th Gen AMD EPYC processors:

- *AMD EPYC™ 9005 Processor Architecture Overview*
- *BIOS & Workload Tuning Guide for AMD EPYC™ 9005 Series Processors*
- *Memory Population Guidelines for AMD EPYC™ 9005 Series Processors*

Chapter 2: General Best Practices for Container Deployment

This chapter recommends the following best practices for container deployment on OpenShift:

- **Increase max-pods for AMD EPYC Processor SKUs that can handle more density.** The default maximum number of pods that can run on a worker node is 250. AMD EPYC processors can potentially handle many more than this depending on container resource sizing. Modify the `maxPods` kubelet flag to take advantage of the compute, memory, and IO density found in AMD EPYC processors.
- **Reserve CPUs and memory resources for kubelet and system daemons and eviction thresholds.** Kubelet services run on every OpenShift node in a cluster. Worker nodes communicate with the Controller node by sending heartbeats every few seconds to determine node availability and pass other health check data. The kubelet service therefore uses CPU resources even when no workload is running. An OpenShift node can be scheduled to capacity, meaning that pods can consume all available capacity and leave few resources for system daemons that power the OS and OpenShift itself. Allocating sufficient resources to pods and the daemons will avoid node resource starvation and resulting competition for resources. Kubelet can use eviction thresholds to proactively terminate pods and reclaim resources. Kubelet monitors node resources such CPU, memory, and filesystem I/O on the nodes and can fail one or more pod(s) on that node when one or more resources reach a specific consumption level, thereby reclaiming resources and preventing starvation. See "Recommended Settings" on page 5.
- **Reducing noisy neighbor scenarios:** Each container has unlimited access to host CPU cycle by default, which means that a "noisy neighbor" container can drain resources from another container on the same host. You can mitigate this problem by limiting the CPU, memory, and/or I/O resources available to each container in the pod definition file. Defining container resource requests and limits determines the Quality of Service (QoS) class of the pod. OpenShift automatically classifies pods into one of three QoS categories:
  - **Guaranteed:** Pods receive this classification when the container in the pod has a CPU and memory request and limit. The requests and limits must be equal.
  - **Burstable:** Pods receive this classification when it does not meet the **Guaranteed** QoS class and has a container with either a memory or CPU request.
  - **BestEffort:** Pods receive this classification if the container has no CPU or memory request or limit. **BestEffort** pods have the lowest scheduling priority and could be evicted to make room for **Guaranteed** or **Burstable** pods.

  On a SMT enabled node, kubelet treats physical and logical cores with the same scheduling priority. In some cases, pods could be assigned virtual and physical cores that are not siblings, which can cause different containers to share a physical core and contribute to a noisy neighbor problem. Setting the OpenShift `cpu-manager-policy-options` static policy option flag to `full-pcpus-only=true` will always allocate full physical cores to prevent SMT thread misalignment. Kubelet will only admit pods if the entire CPU request for all containers can be fulfilled by allocating full physical cores. For example, kubelet will not admit a **Guaranteed** pod with 1 CPU in an SMT environment. The result will generate a SMTAlignmentError message. `full-pcpus-only` focuses on physical core assignments only; it does not have CCX awareness.

  The AMD EPYC chiplet architecture features a split L3 cache topology, where a group of 8 or 16 CPU or vCPU cores (16 or 32 threads when SMT is enabled, respectively) shares the same L3 cache or Core Complex Die (CCD). Each die holds multiple CCDs. In a split L3 topology, a noisy neighbor can occur in the L3 cache, which can greatly impact workloads that are sensitive to cache or memory latency. AMD has mitigated this by contributing to the NRI Topology Aware Plugin*. Deploying this plugin will ensure optimal pod placement for Guaranteed QoS pods, thereby minimizing the number of L3 caches a pod/container shares with other pods/containers.

- **Implement CPU pinning for better performance:** OpenShift uses CFS to enforce pod CPU limits. Again, a container has unlimited access to host CPU cycles by default. A single compute node in an OpenShift cluster can run multiple pods, some of which may be running CPU-intensive workloads. Pods in this scenario might contend for the CPU resources available to that compute node. The workload can move (load balance) to different CPUs when the level of contention rises depending on whether the pod is throttled and the current CPU availability at scheduling time. Some workloads have no problem with this; however, if the system is heavily utilized, then the scheduler may spend more time load balancing than processing the application. Some workloads may be sensitive to context switches. Both scenarios may impact workload performance, and pinning the containers may be helpful. This feature is disabled by default.
- **Implement Node level NUMA topology alignment for CPU and PCI devices:** CPU pinning alone is not enough for latency-sensitive workloads. For example, a network-I/O-intensive workload may be pinned to CCXs or CCDs from different NUMA nodes. By default, OpenShift does not guarantee that a container application will be assigned resources that are local to a NUMA node. The Topology Manager provides the Hint Provider interface for OpenShift components to send and receive topology information. This acts as a source of truth that allows other kubelet components to make topology-aligned resource allocation choices. Topology Manager aligns the resources requested by Hint Provider so as to assign CPU and I/O to the container or pod from the same NUMA node. The device plugin must leverage the OpenShift Device Plugin API and send back TopologyInfo struct as part of the device registration. The Topology Manager provides two distinct settings:
    - **Scope:** Aligns resources at the pod or container level.
    - **Policy:** Defines how the alignment is carried out, which will be `best-effort`, `restricted`, or `single-numa-node`.
- **Use NUMA-Aware Memory Manager for platforms with NUMA nodes greater than one:** Get the best performance and latency for your workload by aligning container CPUs, peripheral devices, and memory to the same NUMA locality. OpenShift-based versions prior to Kubernetes v1.22 included a kubelet with a NUMA topology manager that aligned CPUs and PCI devices, but not memory. The Linux kernel made best-effort attempts to allocate memory for tasks from the same NUMA node where the executing container was placed but could not guarantee that placement.

Chapter 3: Recommended Settings

OpenShift provides a few tuning knobs to optimize containerized workload deployment. These settings affect scheduler behavior of Guaranteed QoS pods. AMD recommended using Guaranteed QoS pods for production deployments. The following sections describe how to enable those tuning knobs and other settings that assist with node stability. These settings are all associated with the kubelet service and modifications are contained in one file.

3.1 - Reserving CPU for Kubelet and System Daemons

`reservedSystemCPUs` is a kubelet flag that defines an explicit CPU set for the OS system daemons. To enable this flag:

1. Label the worker `MachineConfigPool` with a `custom-kubelet:` label. For example:
   `# oc edit machineconfigpool worker`
2. Append a `custom-kubelet` to the labels field.
3. Create a `KubeletConfig` custom resource yaml file. Refer to the `kubelet-tuning.yaml` file example below and ensure the `custom-kubelet` label matches the one created in the previous step.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: epyc-tuning
spec:
  machineConfigPoolSelector:
    matchLabels:
      custom-kubelet: kubelet-tuning
  kubeletConfig:
     reservedSystemCPUs: 0-7,256-263
```

4. Create the dynamic KubeletConfig custom resource:
   `# oc create -f kubelet-tuning.yaml`

3.2 - Container Pinning Settings

3.2.1 - Enable OpenShift Static CPU Manager Policy

By default, the scheduler load balances such that the containerized application will bounce around to different CPUs. Set CPU Manager to static to pin the pods or containers.

1. Append the option `cpuManagerPolicy: static` to the `kubeletConfig:` parameter of the **KubeletConfig** custom resource yaml file.
2. You may modify and redeploy the previous KubeletConfig custom resource or create a new one.
   ```
   # oc create -f <kubeletConfig-cpuManagerPolicy>.yaml
   ```

Only Guaranteed QoS pods will be pinned to CPUs when the Static CPU Manager Policy is enabled.

3.3 - CPU Resource Management for Optimal Pod Placement

A base OpenShift deployment is not aware of the AMD EPYC 9005 architecture that groups CPUs into CCDs that share the same L3, as described in "General Best Practices for Container Deployment" on page 3. The NRI Topology Aware Plugin* applies hardware-aware resource allocation policies to Guaranteed QoS pods on a node. This policy searches for pools of allocatable CPUs based on sockets, NUMAs, and CCDs available on the node. For example, if pod CPU request & limits can fit within a CCD and there are enough available CPUs within that pool, then the policy will assign CPU resources from that pool. If the pod cannot fit within a CCD, then the policy will minimize CPU assignment across multiple CCDs. Please see Figures 3-1, 3-2, and 3-3 for examples of pod placement using the NRI plugin.

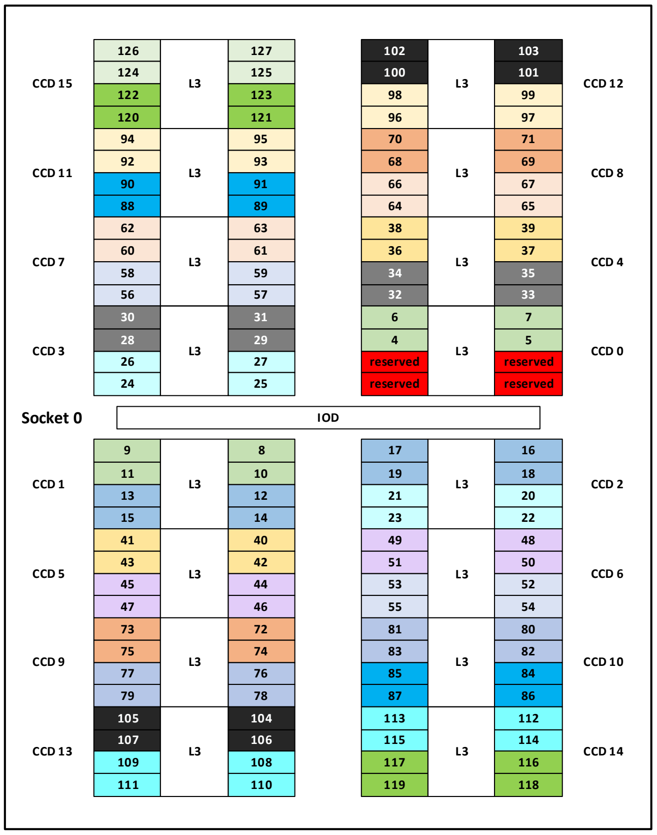
Prerequisites:

- OpenShift 4.13+
- Helm3.0.0+

You must have the following Open Container Initiative (OCI) compatible container engines to support the Node Resource Interface (NRI) framework that defines the infrastructure for runtime plugin extensions in order to use the Topology-Aware Policy.

- containerd 1.7.0+
- CRI-O v1.26.0+

Please see NRI Plugins* for installation instructions and Configuration options* for additional plugin configuration options.


Socket 0
IOD
L3
CCD 15 126 127 124 125 122 123 120 121
CCD 12 102 103 100 101 98 99 96 97
CCD 11 94 95 92 93 90 91 88 89
CCD 8 70 71 68 69 66 67 64 65
CCD 7 62 63 60 61 58 59 56 57
CCD 4 38 39 36 37 34 35 32 33
CCD 3 30 31 28 29 26 27 24 25
CCD 0 6 7 4 5 reserved reserved reserved reserved
CCD 1 9 8 11 10 13 12 15 14
CCD 2 17 16 19 18 21 20 23 22
CCD 5 41 40 43 42 45 44 47 46
CCD 6 49 48 51 50 53 52 55 54
CCD 9 73 72 75 74 77 76 79 78
CCD 10 81 80 83 82 85 84 87 86
CCD 13 105 104 107 106 109 108 111 110
CCD 14 113 112 115 114 117 116 119 118

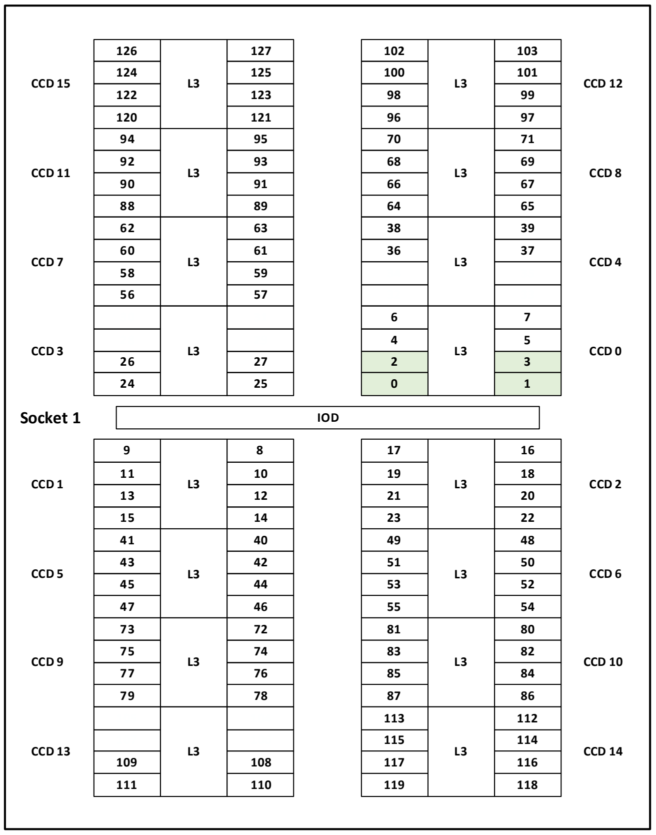

Socket 1
IOD
L3
CCD 15 126 127 124 125 122 123 120 121
CCD 12 102 103 100 101 98 99 96 97
CCD 11 94 95 92 93 90 91 88 89
CCD 8 70 71 68 69 66 67 64 65
CCD 7 62 63 60 61 58 59 56 57
CCD 4 38 39 36 37
CCD 3 26 27 24 25
CCD 0 6 7 4 5 2 3 0 1
CCD 1 9 8 11 10 13 12 15 14
CCD 2 17 16 19 18 21 20 23 22
CCD 5 41 40 43 42 45 44 47 46
CCD 6 49 48 51 50 53 52 55 54
CCD 9 73 72 75 74 77 76 79 78
CCD 10 81 80 83 82 85 84 87 86
CCD 13 109 108 111 110
CCD 14 113 112 115 114 117 116 119 118

Figure 3-1: Case 1: OCP Static CPU Manager policy enabled and CCD misaligned

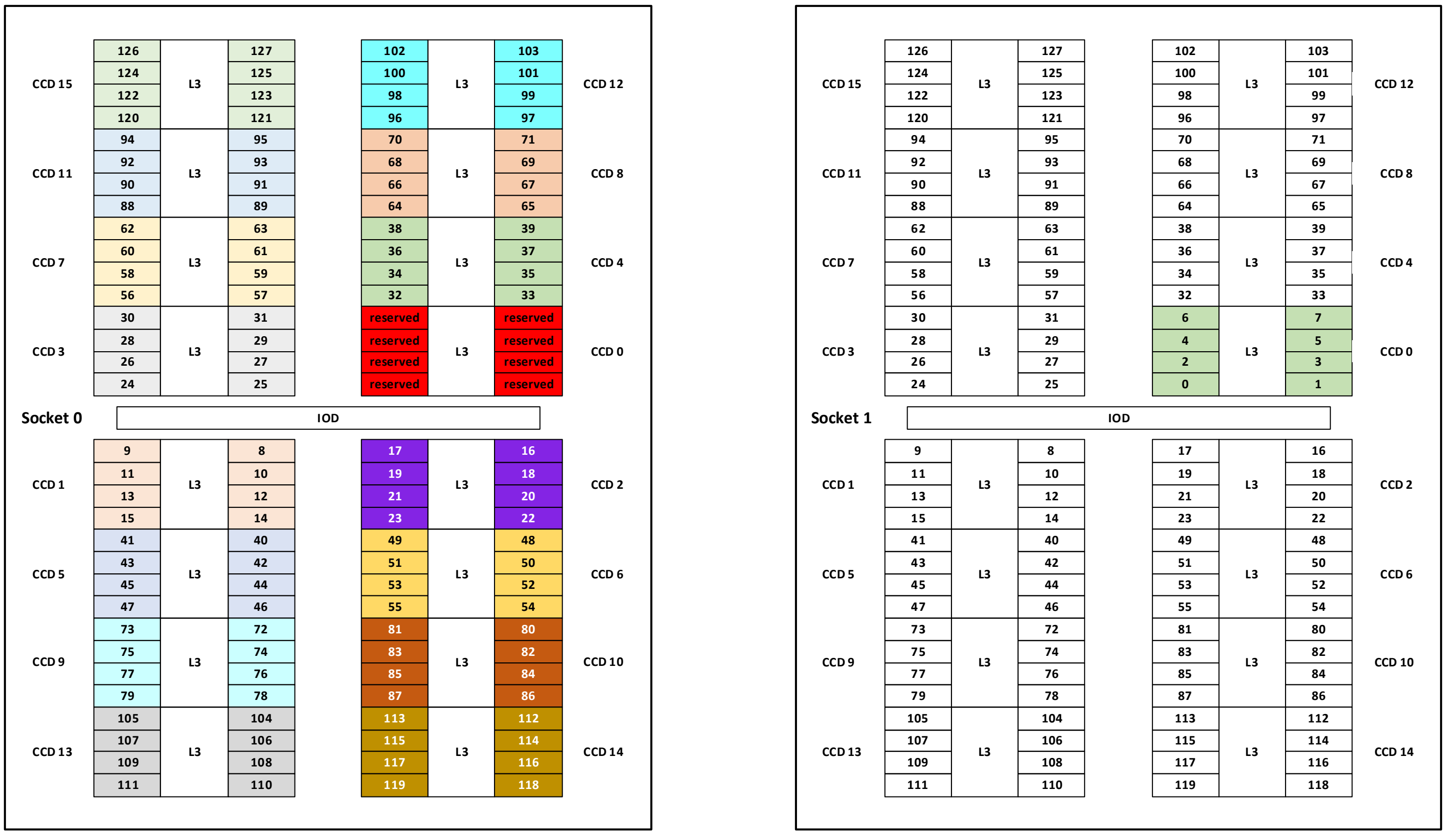


Figure 3-2: Case 2: OCP Static CPU Manager policy enabled and CCD aligned by reserving full CCD for --reserved-cpus

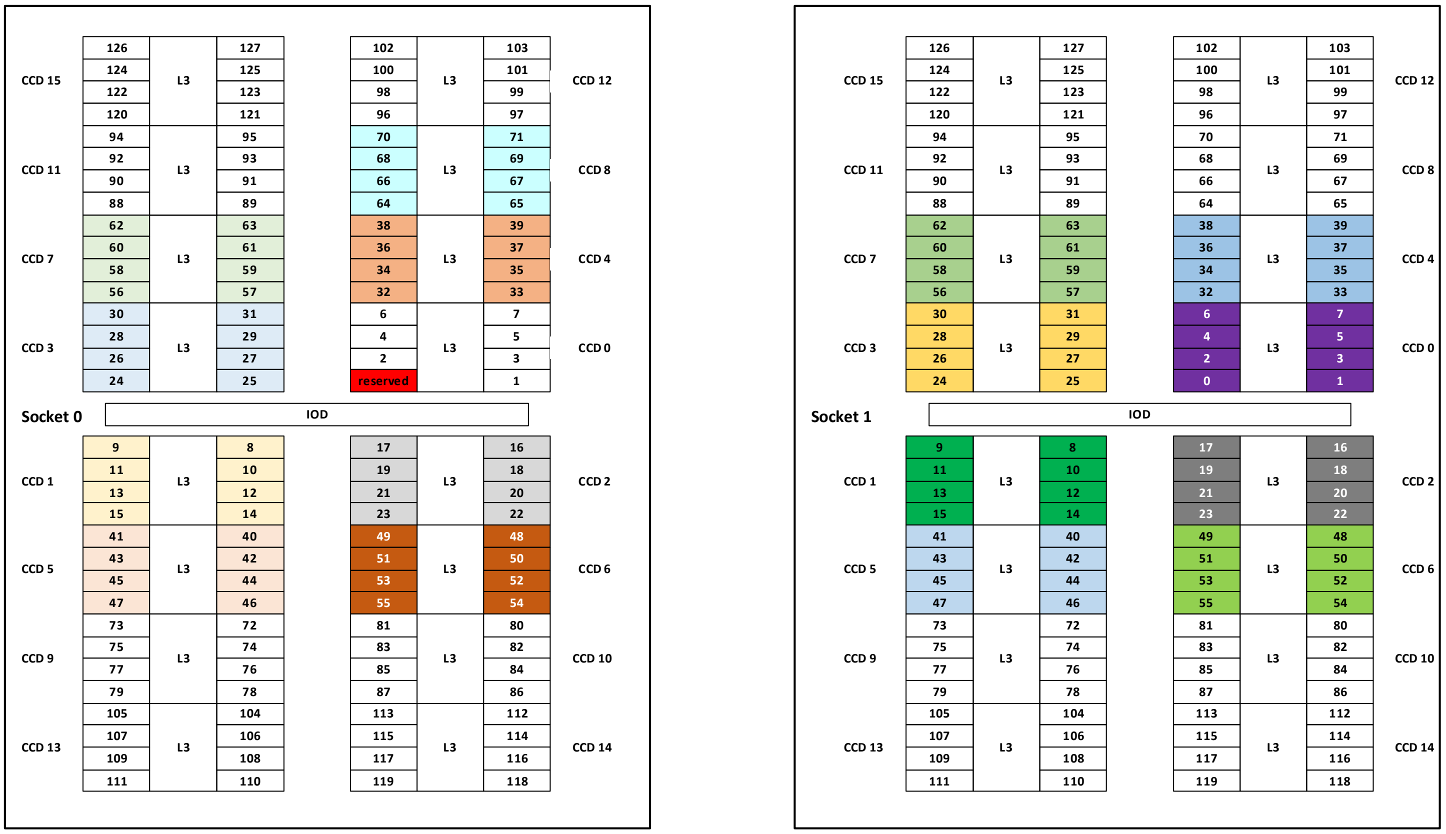


Figure 3-3: Case 3: NRI Topology-Aware policy enabled

Figure 3-4 shows a sample performance uplift for a TPROC-C benchmark deployed on a OpenShift node with the `static-cpu-manager` policy versus the NRI plugin. This information is provided for reference only; your actual performance may vary widely due to system configuration, workload, and other variables.

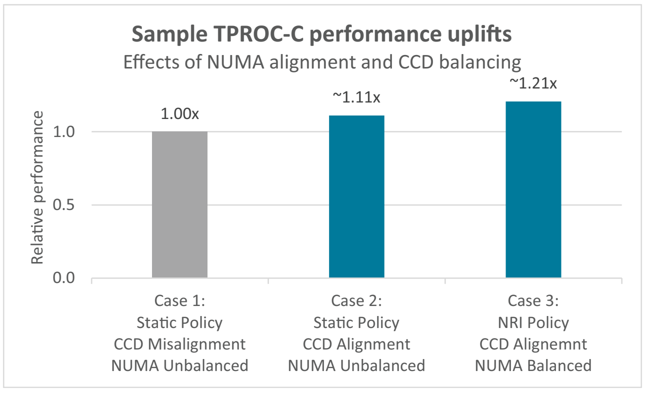


Figure 3-4: HammerDB TPROC-C performance enhancement using the topology-aware policy

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 4: Resources

- What is Red Hat OpenShift*
- Sizing OpenShift Clusters and Nodes*
- Reserve Compute Resources for System Daemons*
- Recommended control plane practices*

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 5: Glossary

- **OCP:** Shorthand for OpenShift.
- **Pod:** Smallest unit of work known to OpenShift. Pod can contain one of more containers. Node – Worker machine within the OCP cluster.
- **QoS:** Quality of Service.
- **CFS:** Completely Fair Scheduler.
- **SMT:** Simultaneous Multiple Threading.

OpenShift® Container Tuning Guide for AMD EPYC™ 9005 Processors

PID: 58724

Sonemaly Phrasavath