AMD EPYC

TUNING GUIDE
AMD EPYC 9004

Hadoop®

Publication	58013
Revision	1.3
Issue Date	June, 2023

Date	Version	Changes
July, 2022	0.1	Initial NDA partner release
Sep, 2022	0.2	Significant global updates
Nov, 2022	1.0	Initial public release
Dec, 2022	1.1	Minor errata corrections
Mar, 2023	1.2	Added 97xx OPN and AMD 3D V-Cache™ technology information
Jun, 2023	1.3	Second public release

Audience

This document is intended for a technical audience such as Hadoop® application architects, production deployment, and performance engineering teams with:

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

Authors

Bryon Georgson, Jiacong He, Dinesh Chitlangia, and Gnanakumar Rajaram

Note: All of the settings described in this Tuning Guide apply to all AMD EPYC 9004 Series Processors of all core counts with or without AMD 3D V-Cache™ except where explicitly noted otherwise.

Table of Contents

Chapter 1 Introduction

The Apache™ Hadoop® software library is a framework that allows distributed processing of large datasets across clusters of computers using simple programming models. It scales from single servers to thousands of machines that can each offer local compute, storage, or both. Hadoop uses software techniques to deliver high availability instead of relying on hardware; implementing Hadoop in a cluster leverages the software's built-in resiliency to deliver high availability by tolerating failures from a failed drive to an entirely failed server. Hadoop runs across such a wide range of diverse applications and environments that no golden rule exists for tuning the cluster. Different Hadoop distribution vendors expose different settings through their management software. This guide recommends parameter settings that will have the most impact on Hadoop cluster performance across the following categories:

- Basic Input Output System (BIOS)
- Linux™ OS
- Hadoop Distributed File System (HDFS)
- Yet Another Resource Manager (YARN)

The AMD testing performed for this Tuning Guide used Cloudera CDH, but they should apply to Hadoop 3.x-based distributions, including Cloudera, Cloudera Hortonworks Data Platform (formerly Hortonworks), and HPE Ezmeral Data Fabric (formerly MapR).

1.1 Hadoop Tuning Challenges

Hadoop is a sophisticated software framework that consists of several components interacting with each other across multiple systems. Bottlenecks in any of these components can and will cause poor Hadoop workload performance. Every system element from BIOS settings, OS settings, disk I/O, and networking affects Hadoop performance.

The Hadoop stack itself has several configuration settings that can impact performance and therefore require a certain degree of Hadoop ecosystem familiarity to optimize. Changing one setting can influence other settings, making Hadoop tuning an iterative process where one must know the implications of a particular settings change before continuing with additional tuning.

This page intentionally left blank.

Chapter 2

AMD EPYC™ 9004 Series Processors

AMD EPYC™ 9004 Series Processors represent the fourth generation of AMD EPYC server-class processors. This generation of AMD EPYC processors feature AMD's latest "Zen 4" based compute cores, next-generation Infinity Fabric, next-generation memory & I/O technology, and use the new SP5 socket/packaging.

2.1 General Specifications

AMD EPYC 9004 Series Processors offer a variety of configurations with varying numbers of cores, Thermal Design Points (TDPs), frequencies, cache sizes, etc. that complement AMD's existing server portfolio with further improvements to performance, power efficiency, and value. Table 1-1 lists the features common to all AMD EPYC 9004 Series Processors.

Common Features of all AMD EPYC 9004 Series Processors	
Compute cores	Zen4-based
Core process technology	5nm
Maximum cores per Core Complex (CCX)	8
Max memory per socket	6 TB
Max # of memory channels	12 DDR5
Max memory speed	4800 MT/s DDR5
Max lanes Compute eXpress Links	64 lanes CXL 1.1+
Max lanes Peripheral Component Interconnect	128 lanes PCIe® Gen 5

Table 2-1: Common features of all AMD EPYC 9004 Series Processors

2.2 Model-Specific Features

Different models of 4th Gen AMD EPYC processors have different feature sets, as shown in Table 1-2.

AMD EPYC 9004 Series Processor (Family 19h) Features by Model		
Codename	"Genoa"*	"Bergamo"*
Model #	91xx-96xx	97xx
Max number of Core Complex Dies (CCDs)	12	8
Number of Core Complexes (CCXs) per CCD	1	2
Max number of cores (threads)	96 (192)	128 (256)
Max L3 cache size (per CCX)	1,152 MB (96 MB)♦	256 MB (16 MB)
Max Processor Frequency	4.4 GHz♦♦	3.15 GHz
Includes ♦AMD 3D V-Cache (9xx4X) and ♦♦high-frequency (9xx4F) models.		
*GD-122: The information contained herein is for informational purposes only and is subject to change without notice. Timelines, roadmaps, and/or product release dates shown herein and plans only and subject to change. "Genoa" and "Bergamo" are codenames for AMD architectures and are not product names.		

Table 2-2: AMD EPYC 9004 Series Processors features by model

2.3 Operating Systems

AMD recommends using the latest available targeted OS version and updates. Please see AMD EPYC™ Processors Minimum Operating System (OS) Versions for detailed OS version information.

2.4 Processor Layout

AMD EPYC 9004 Series Processors incorporate compute cores, memory controllers, I/O controllers, RAS (Reliability, Availability, and Serviceability), and security features into an integrated System on a Chip (SoC). The AMD EPYC 9004 Series Processor retains the proven Multi-Chip Module (MCM) Chiplet architecture of prior successful AMD EPYC processors while making further improvements to the SoC components.

The SoC includes the Core Complex Dies (CCDs), which contain Core Complexes (CCXs), which contain the "Zen 4"-based cores. The CCDs surround the central high-speed I/O Die (and interconnect via the Infinity Fabric). The following sections describe each of these components.

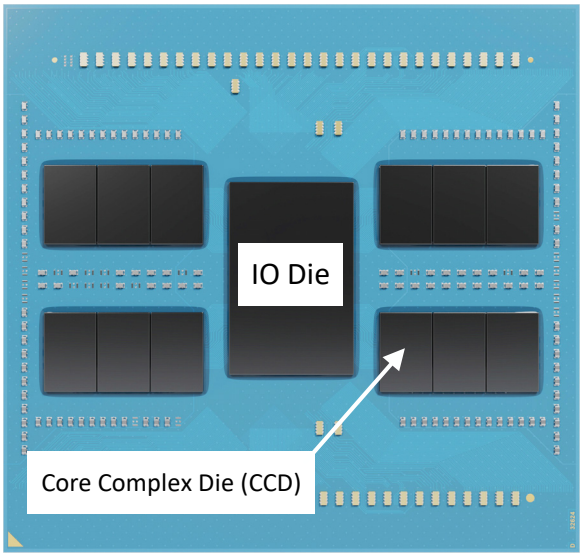

Figure 2-1: AMD EPYC 9004 configuration with 12 Core Complex Dies (CCD) surrounding a central I/O Die (IOD)

2.5 "Zen 4" Core

AMD EPYC 9004 Series Processors are based on the new "Zen 4" compute core. The "Zen 4" core is manufactured using a 5nm process and is designed to provide an Instructions per Cycle (IPC) uplift and frequency improvements over prior generation "Zen" cores. Each core has a larger L2 cache and improved cache effectiveness over the prior generation. Each "Zen 4" core includes:

- Up to 32 KB of 8-way L1 I-cache and 32 KB of 8-way of L1 D-cache
- Up to a 1 MB private unified (Instruction/Data) L2 cache.

Each core supports Simultaneous Multithreading (SMT), which allows 2 separate hardware threads to run independently, sharing the corresponding core's L2 cache.

2.6 Core Complex (CCX)

Figure 2-2 shows a Core Complex (CCX) where up to eight "Zen 4"-based cores share a L3 or Last Level Cache (LLC). Enabling Simultaneous Multithreading (SMT) allows a single CCX to support up to 16 concurrent hardware threads.

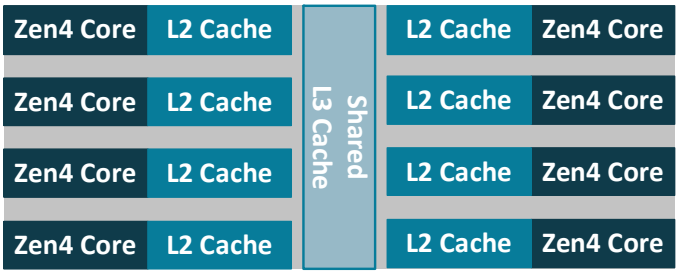

Figure 2-2: Top view of 8 compute cores sharing an L3 cache (91xx-96xx models)

2.7 Core Complex Dies (CCDs)

The Core Complex Die (CCD) in an AMD EPYC 9xx4 Series Processor may contain either one or two CCXs, depending on the processor (91xx-96xx "Genoa" vs. 97xx "Bergamo"), as shown in Figure 2-5.

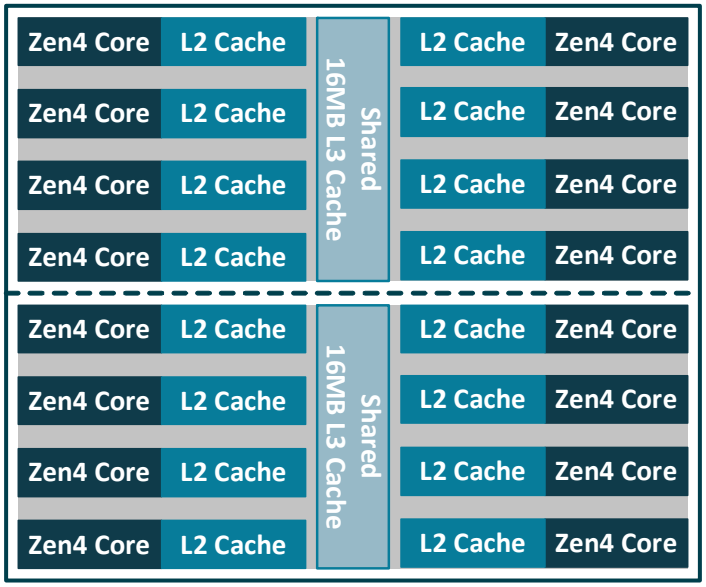

Figure 2-3: 2 CCXs in a single 4th Gen AMD EPYC 97xx CCD

Each of the Core Complex Dies (CCDs) in a 97xx model AMD EPYC 9004 Series Processor contains two CCXs (Figure 2-5):

AMD EPYC 9004 Series Processor	91xx-96xx	97xx
# of CCXs within a CCD	1	2

Table 2-3: CCXs per CCD by AMD EPYC model

You can disable cores in BIOS using one or both of the following approaches:

- Reduce the cores per L3 from 8 down to 7,6,5,4,3,2, or 1 while keeping the number of CCDs constant. This approach increases the effective cache per core ratio but reduces the number of cores sharing the cache.
- Reduce the number of active CCDs while keeping the cores per CCD constant. This approach maintains the advantages of cache sharing between the cores while maintaining the same cache per core ratio.

2.8 AMD 3D V-Cache™ Technology

AMD EPYC 9xx4X Series Processors include AMD 3D V-Cache™ die stacking technology that enables 97xx to achieve more efficient chiplet integration. AMD 3D Chiplet architecture stacks L3 cache tiles vertically to provide up to 96MB of L3 cache per die (and up to 1 GB L3 Cache per socket) while still providing socket compatibility with all AMD EPYC™ 9004 Series Processor models.

AMD EPYC 9004 Series Processors with AMD 3D V-Cache technology employ industry-leading logic stacking based on copper-to-copper hybrid bonding "bumpless" chip-on-wafer process to enable over 200X the interconnect densities of current 2D technologies (and over 15X the interconnect densities of other 3D technologies using solder bumps), which translates to lower latency, higher bandwidth, and greater power and thermal efficiencies.

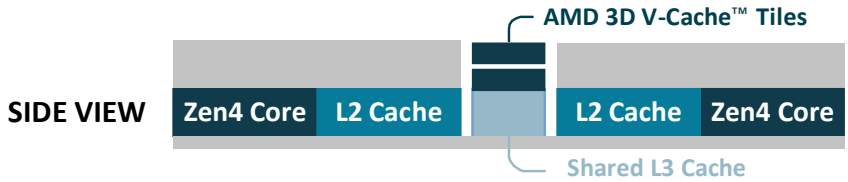

Figure 2-4: Side view of vertically-stacked central L3 SRAM tiles

AMD EPYC 9004 Series Processors	9xx4	9004X (with 3D V-Cache)
Max Shared L3 Cache per CCD	32 MB	96 MB

Table 2-4: L3 cache by processor model

Different OPNs also may have different numbers of cores within the CCX. However, for any given part, all CCXs will always contain the same number of cores.

2.9 I/O Die (Infinity Fabric™)

The CCDs connect to memory, I/O, and each other through an updated I/O Die (IOD). This central AMD Infinity Fabric™ provides the data path and control support to interconnect CCXs, memory, and I/O. Each CCD connects to the IOD via a dedicated high-speed Global Memory Interconnect (GMI) link. The IOD helps maintain cache coherency and additionally provides the interface to extend the data fabric to a potential second processor via its xGMI, or G-links. AMD EPYC 9004 Series Processors support up to 4 xGMI (or G-links) with speeds up to 32Gbps. The IOD exposes DDR5 memory channels, PCIe® Gen5, CXL 1.1+, and Infinity Fabric links.

All dies (chiplets) interconnect with each other via AMD Infinity Fabric technology. Figure 2-6 (which corresponds to Figure 2-2, above) shows the layout of a 96-core AMD EPYC 9654 processor. The AMD EPYC 9654 has 12 CCDs, with each CCD connecting to the IOD via its own GMI connection.

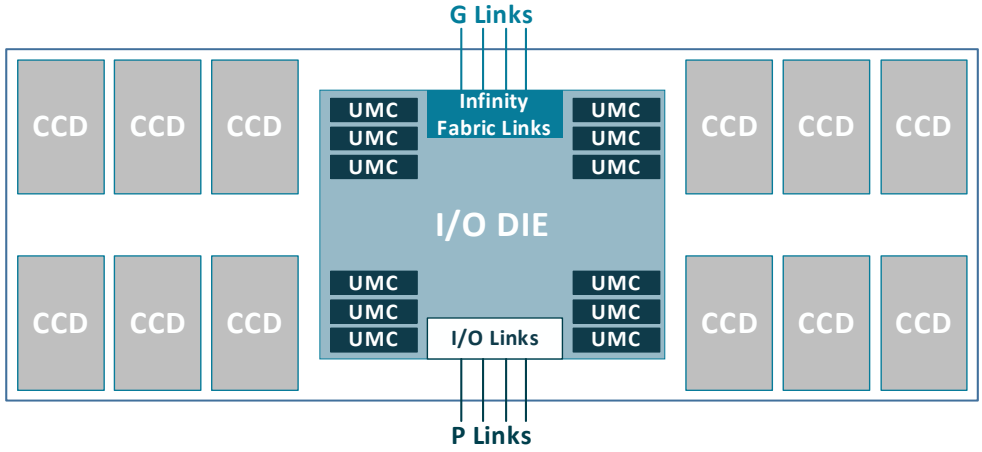

Figure 2-5: AMD EPYC 9654 processor internals interconnect via AMD Infinity Fabric (12 CCD processor shown)

AMD also provides "wide" OPNs (e.g. AMD EPYC 9334) where each CCD connects to two GMI3 interfaces, thereby allowing double the Core-to-I/O die bandwidth.

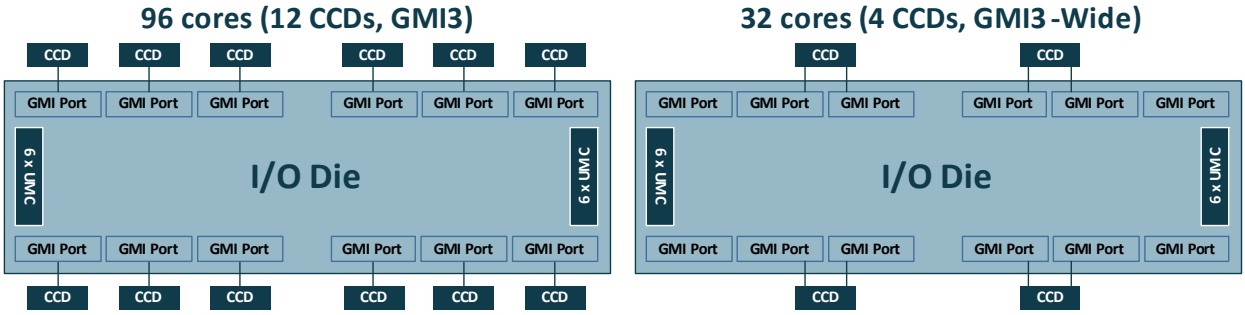

Figure 2-6: Standard vs. Wide GMI links

The IOD provides twelve Unified Memory Controllers (UMCs) that support DDR5 memory. The IOD also presents 4 'P-links' that the system OEM/designer can configure to support various I/O interfaces, such as PCIe Gen5, and/or CXL 1.1+.

2.10 Memory and I/O

Each UMC can support up to 2 DIMMs per channel (DPC) for a maximum of 24 DIMMs per socket. OEM server configurations may allow either 1 DIMM per channel or 2 DIMMs per channel. 4th Gen AMD EPYC processors can support up to 6TB of DDR5 memory per socket. Having additional and faster memory channels compared to previous generations of AMD EPYC processors provides additional memory bandwidth to feed high-core-count processors. Memory interleaving on 2, 4, 6, 8, 10, and 12 channels helps optimize for a variety of workloads and memory configurations.

Each processor may have a set of 4 P-links and 4 G-links. An OEM motherboard design can use a G-link to either connect to a second 4th Gen AMD EPYC processor or to provide additional PCIe Gen5 lanes. 4th Gen AMD EPYC processors support up to eight sets of x16-bit I/O lanes, that is, 128 lanes of high-speed PCIe Gen5 in single-socket platforms and up to 160 lanes in dual-socket platforms. Further, OEMs may either configure 32 of these 128 lanes as SATA lanes and/or configure 64 lanes as CXL 1.1+. In summary, these links can support:

- Up to 4 G-links of AMD Infinity Fabric connectivity for 2P designs.
- Up to 8 x16 bit or 128 lanes of PCIe Gen 5 connectivity to peripherals in 1P designs (and up to 160 lanes in 2-socket designs).
- Up to 64 lanes (4 P-links) that can be dedicated to Compute Express Link (CXL) 1.1+ connectivity to extended memory.
- Up to 32 I/O lanes that can be configured as SATA disk controllers.

2.11 Visualizing AMD EPYC 9004 Series Processors (Family 19h)

This section depicts AMD EPYC 9004 Series Processors that have been set up with four nodes per socket (NPS=4). Please see "NUMA Topology" on page 10 for more information about nodes.

2.11.1 Models 91xx-96xx ("Genoa")

4th Gen AMD EPYC 9004 processors with model numbers 91xx-96xx have up to 12 CCDs that each contain a single CCX, as shown below.

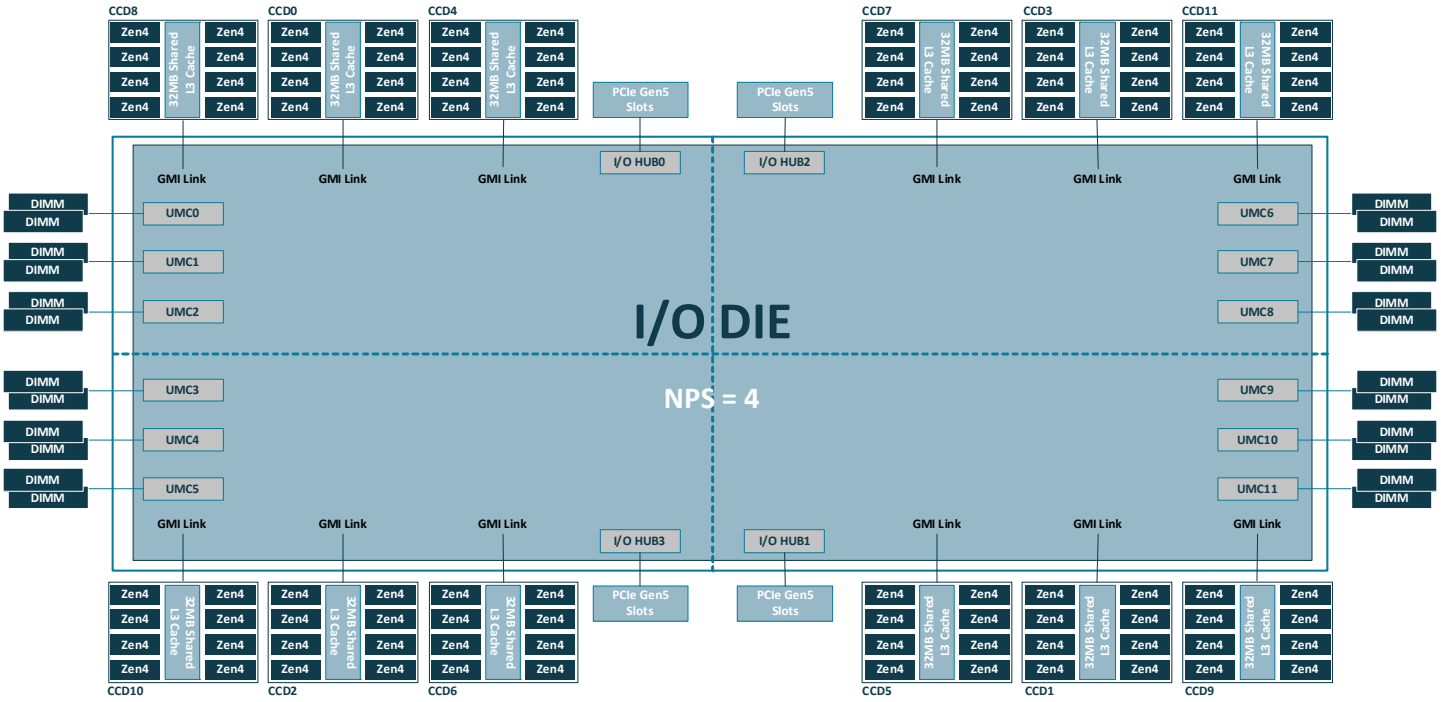

Figure 2-7: The AMD EPYC 9004 SoC consists of up to 12 CCDs and a central IOD for 91xx-96xx models, including "X" OPNs

2.11.2 Models 97xx ("Bergamo")

97xx 4th Gen AMD EPYC 9004 Series Processors with model numbers 97xx have up to 8 CCDs that each contain two CCXs, as shown below.

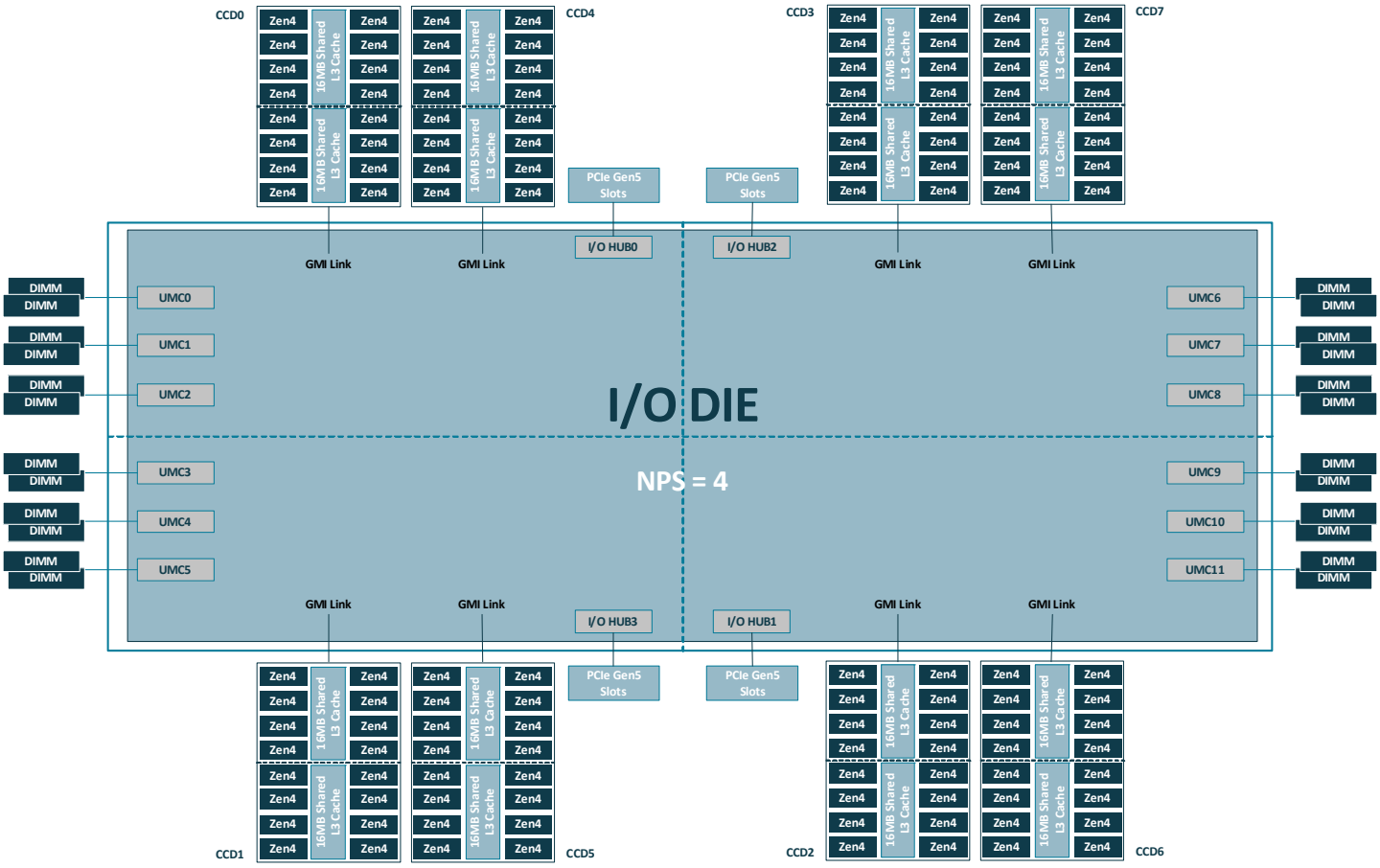

Figure 2-8: The AMD EPYC 9004 System on Chip (SoC) consists of up to 8 CCDs and a central IOD for 97xx models

2.12 NUMA Topology

AMD EPYC 9004 Series Processors use a Non-Uniform Memory Access (NUMA) architecture where different latencies may exist depending on the proximity of a processor core to memory and I/O controllers. Using resources within the same NUMA node provides uniform good performance, while using resources in differing nodes increases latencies.

2.12.1 NUMA Settings

A user can adjust the system **NUMA Nodes Per Socket** (NPS) BIOS setting to optimize this NUMA topology for their specific operating environment and workload. For example, setting NPS=4 as shown in "Memory and I/O" on page 8 divides the processor into quadrants, where each quadrant has 3 CCDs, 3 UMCs, and 1 I/O Hub. The closest processor-memory I/O distance is between the cores, memory, and I/O peripherals within the same quadrant. The furthest distance is between a core and memory controller or IO hub in cross- diagonal quadrants (or the other processor in a 2P configuration). The locality of cores, memory, and IO hub/devices in a NUMA-based system is an important factor when tuning for performance.

The NPS setting also controls the interleave pattern of the memory channels within the NUMA Node. Each memory channel within a given NUMA node is interleaved. The number of channels interleaved decreases as the NPS setting gets more granular. For example:

- A setting of NPS=4 partitions the processor into four NUMA nodes per socket with each logical quadrant configured as its own NUMA domain. Memory is interleaved across the memory channels associated with each quadrant. PCIe devices will be local to one of the four processor NUMA domains, depending on the IOD quadrant that has the corresponding PCIe root complex for that device.

- A setting of NPS=2 configures each processor into two NUMA domains that groups half of the cores and half of the memory channels into one NUMA domain, and the remaining cores and memory channels into a second NUMA domain. Memory is interleaved across the six memory channels in each NUMA domain. PCIe devices will be local to one of the two NUMA nodes depending on the half that has the PCIe root complex for that device.

- A setting of NPS=1 indicates a single NUMA node per socket. This setting configures all memory channels on the processor into a single NUMA node. All processor cores, all attached memory, and all PCIe devices connected to the SoC are in that one NUMA node. Memory is interleaved across all memory channels on the processor into a single address space.

- A setting of NPS=0 indicates a single NUMA domain of the entire system (across both sockets in a two-socket configuration). This setting configures all memory channels on the system into a single NUMA node. Memory is interleaved across all memory channels on the system into a single address space. All processor cores across all sockets, all attached memory, and all PCIe devices connected to either processor are in that single NUMA domain.

You may also be able to further improve the performance of certain environments by using the **LLC (L3 Cache) as NUMA** BIOS setting to associate workloads to compute cores that all share a single LLC. Enabling this setting equates each shared L3 or CCX to a separate NUMA node, as a unique L3 cache per CCD. A single AMD EPYC 9004 Series Processor with 12 CCDs can have up to 12 NUMA nodes when this setting is enabled.

Thus, a single EPYC 9004 Series Processor may support a variety of NUMA configurations ranging from one to twelve NUMA nodes per socket.

Note: If software needs to understand NUMA topology or core enumeration, it is imperative to use documented Operating System (OS) APIs, well-defined interfaces, and commands. Do not rely on past assumptions about settings such as APICID or CCX ordering.

2.13 Dual-Socket Configurations

AMD EPYC 9004 Series Processors support single- or dual-socket system configurations. Processors with a 'P' suffix in their name are optimized for single-socket configurations (see the "Processor Identification" chapter) only. Dual-socket configurations require both processors to be identical. You cannot use two different processor Ordering Part Numbers (OPNs) in a single dual-socket system.

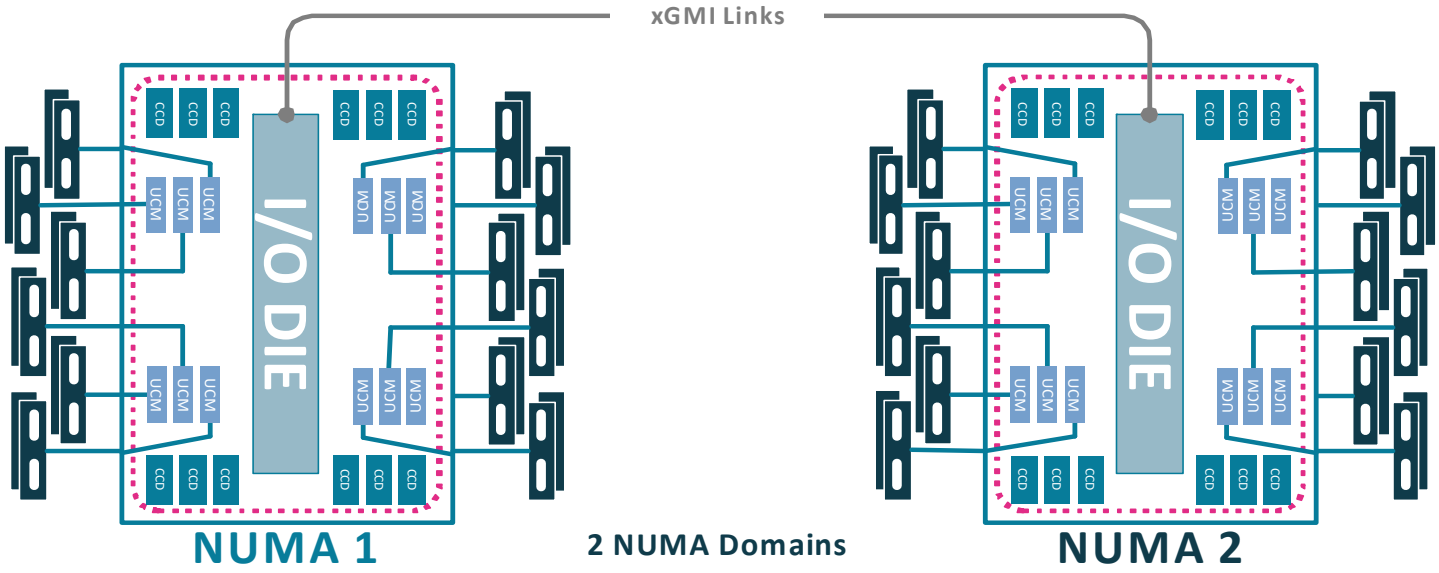

Figure 2-9: Two EPYC 9004 Processors connect through 4 xGMI links (NPS1)

In dual-socket systems, two identical EPYC 9004 series SoCs are connected via their corresponding External Global Memory Interconnect [xGMI] links. This creates a high bandwidth, low latency interconnect between the two processors. System manufacturers can elect to use either 3 or 4 of these Infinity Fabric links depending upon I/O and bandwidth system design objectives.

The Infinity Fabric links utilize the same physical connections as the PCIe lanes on the system. Each link uses up to 16 PCIe lanes. A typical dual socket system will reconfigure 64 PCIe lanes (4 links) from each socket for Infinity Fabric connections. This leaves each socket with 64 remaining PCIe lanes, meaning that the system has a total of 128 PCIe lanes. In some cases, a system designer may want to expose more PCIe lanes for the system by reducing the number of Infinity Fabric G-Links from 4 to 3. In these cases, the designer may allocate up to 160 lanes for PCIe (80 per socket) by utilizing only 48 lanes per socket for Infinity Fabric links instead of 64.

A dual-socket system has a total of 24 memory channels, or 12 per socket. Different OPNs can be configured to support a variety of NUMA domains.

Chapter

3

BIOS Defaults Summary

This chapter provides high-level lists of the default AMD EPYC 9004 BIOS settings and their default values. Please see Chapter 4 of the *BIOS & Workload Tuning Guide for AMD EPYC™ 9004 Series Processors* (available from AMD EPYC Tuning Guides) for detailed descriptions. Later chapters in this Tuning Guide discuss the BIOS options as they relate to a specific workload or set of workloads.

Note: The default setting names and values described in this chapter are the AMD default names and values that serve as recommendations for OEMs. End users must confirm their OEM BIOS setting availability and options.

AMD strongly recommends that customers download and install the latest BIOS update for your AMD EPYC 9004 Series Processor-based server from your platform vendor. BIOS updates often help customers by providing new and updated features, bug fixes, enhancements, security features, and other improvements. These improvements can help your system software stability and dependency modules (such as hardware, firmware, drivers, and software) by giving you a more robust environment to run your applications.

3.1 Processor Core Settings

Name	Default	Description
SMT Control	Auto	• **Enabled/Auto:** Two hardware threads per core. • **Disabled:** Single hardware thread per core.
L1 Stream HW Prefetcher	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
L1 Stride Prefetcher	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
L1 Region Prefetcher	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
L1 Burst Prefetch Mode	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
L2 Stream HW Prefetcher	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
L2 Up/Down Prefetcher	Auto	• **Enabled/Auto:** Enables the prefetcher. • **Disabled:** Disables the prefetcher.
Core Performance Boost	Auto	• **Enabled/Auto:** Enables Core Performance Boost. • **Disabled:** Disables Core Performance Boost.
BoostFmaxEn	Auto	• **Auto:** Use the default Fmax • **Manual:** User can set the boost Fmax
BoostFmax	Auto	Specify the boost Fmax frequency limit to apply to all cores (MHz in decimal)
Global C-State Control	Auto	• **Enabled/Auto:** Controls IO based C-state generation and DF C-states, including core processor C-States • **Disabled:** AMD strongly recommends not disabling this option because this also disables core processor C-States.

Table 3-1: Processor core BIOS settings

X3D	Auto	Enables or disables AMD 3D V-Cache™ technology on Cache Optimized (9004X) processors. • **Auto:** Enabled on an AMD EPYC 9004 Series processor with AMD 3D V-Cache™ technology, enabling this option enables the AMD 3D V-Cache module in the CCD to increase the total size of the L3 cache memory to 96MB • **Disabled:** Disabling this option reduces the L3 cache in the CCD to 32MB. *Note: This option only applies to AMD EPYC 9004 Series Processors with AMD 3D V-Cache technology.* *Note: AMD engineers performed extensive internal testing and validation for various applications using the X3D BIOS option found in AMD EPYC 9xx4X processors with AMD 3D V-Cache technology. This testing and validation cannot cover all applications or use cases. Testing has shown AMD 3D V-Cache to be beneficial for most workloads, however AMD recommends that you test and evaluate the benefits of enabling or disabling the X3D BIOS option for your application workloads in your environment and proceeding based on those results.*

Table 3-1: Processor core BIOS settings

3.2 Power Efficiency Settings

Name	Default	Description
Power Profile Selection	Auto	• **Auto/0:** High-performance mode • **1:** Efficiency mode • **2:** Maximum I/O performance mode
Determinism Control	Auto	• **Auto:** Use default performance determinism settings. • **Manual:** Specify custom performance determinism settings.
Determinism Enable	Auto	• **Auto:** Performance. • **1:** Power.
TDP Control	Auto	• **Auto:** Use platform- and OPN-default TDP. • **Manual:** Set custom configurable TDP.
TDP	OPN Max	This option appears once the user sets the **TDP Control** to **Manual**. • **Values 85-400:** Set configurable TDP, in watts.
PPT Control	Auto	Enables or disables the **PPT** control. • **Auto:** Automatically set PPL in watts. • **Manual:** Specify a custom PPL.
PPT	OPN Max	This option appears once the user sets the **PPT Control** to **Manual**. • **Values 85-400:** Set configurable PPT, in watts.
CPPC	Auto	• **Enabled/Auto:** Allows the OS to make performance/power optimization requests using ACPI CPPC. • **Disabled:** Prevents the OS from making performance/power optimization requests using ACPI CPPC.

Table 3-2: Power efficiency BIOS settings

3.3 NUMA and Memory Settings

Name	Default	Description
LLC as NUMA Domain (ACPI SRAT L3 Cache as NUMA Domain)	Disabled	• **Disabled (recommended):** Both NUMA nodes (`cpubind`) and memory interleaving (`membind`) are determined by the NPS setting. • **Enabled:** Overrides the NPS setting for # of NUMA nodes by mapping each LLC as a NUMA node. This does not impact the memory interleaving
Nodes Per Socket (NPS)	1	**Memory Interleaving:** The **NPS** setting always determines the memory interleaving regardless of whether **LLC as NUMA** is **Enabled** or **Disabled**. # of NUMA nodes (if **LLC as NUMA Domain** is **Disabled**): • **NPS1/Auto:** One NUMA node per socket (Most cloud providers use this as it provides consistent average memory latency to all the accesses within a socket). • **NPS2:** Two NUMA nodes per socket. • **NPS4:** Four NUMA nodes per socket • **NPS0 (not recommended):** Only applicable for dual-socket systems. A single NUMA node is created for the whole two-socket platform. AMD recommends either NPS1 or NPS4 depending on your use case. **Windows systems:** Make sure that the number of logical processors per NUMA node is <=64. You can do this by using NPS2 or NPS4 instead of the default NPS1.
Memory Target Speed	Auto	• **Auto:** Determine the maximum memory speed based on SPD information from populated DIMMs and platform memory speed support. Alternatively, you can select: • **Values 3200–5600 MT/s:** Run the DRAM memory target clock speed at the specified speed. The DRAM memory target is the DDR rate. Your OEM system default value may vary.
Memory Interleaving	Auto	• **Auto/Enable:** Enables memory interleaving. • **Disable:** Allows for disabling memory interleaving. The N**UMA Nodes per Socket** setting will be honored regardless of this setting. AMD strongly recommends not disabling this setting because most production deployments benefit from memory interleaving.

Table 3-3: NUMA and memory BIOS settings

3.4 Infinity Fabric Settings

Name	Default	Description
3-4 xGMI Link Max Speed	Auto	• 12 Gbps • 16 Gbps • 17 Gbps • 18 Gbps • 20 Gbps • 22 Gbps • 23 Gbps • 24 Gbps • 25 Gbps/Auto • 26 Gbps • 27 Gbps • 28 Gbps • 30 Gbps • 32 Gbps Your OEM system default value may vary.
xGMI Link Width Control	Auto	• **Auto:** Use the default xGMI link width controller settings. • **Manual:** Specify a custom xGMI link width controller setting.
xGMI Force Link Width Control	Auto	• **Unforce:** Do not force the xGMI to a fixed width. • **Force:** Use the xGMI link to the user-specified width.
xGMI Force Link Width	Auto	• **0:** Force xGMI link width to x4. • **1:** Force xGMI link width to x8. • **2:** Force xGMI link width to x16.
xGMI Max Link Width Control	Auto	• **Auto:** Use the default xGMI link width controller settings. • **Manual:** Specify a custom xGMI link with controller setting.
xGMI Max Link Width	Auto	• **0:** Set max xGMI link width to x8. • **1:** Set max xGMI link width to x16.
APBDIS	Auto	• **0/Auto:** Dynamically switch the Infinity Fabric P-state based on link usage. • **1:** Enabled fixed Infinity Fabric P-state control.
DfPstate Range Support	Auto	• **Auto:** If this feature is enabled, the range value setting should follow the rule that MaxDfPstate<=MinDfPstate. Otherwise, it will not work. • **Enable:** Add the values MaxDfPstate & MinDfPstate. • **Disable:** No MaxDfPstate & MinDfPstate option.

Table 3-4: Infinity Fabric BIOS settings

DF C-States	Auto	Controls DF C-states. • **Disabled:** Prevents the AMD Infinity Fabric from entering a low-power state. • **Enabled/Auto:** Allows the AMD Infinity Fabric to enter a low-power state.

Table 3-4: Infinity Fabric BIOS settings

3.5 PCIe, I/O, Security, and Virtualization Settings

Name	Default	Description
Local APIC Mode	Auto(0x02)	• **xAPIC:** Use xAPIC, supports up to 255 cores. • **x2APIC:** Supports more than 255 cores. • **Auto:** The system will choose the mode that best fits the number of active cores in the system. • **Compatibility:** Threads below 255 run in xAPIC with xAPIC ACPI structures, and threads 255 and above run in x2 mode with x2 ACPI structures. • **XApicMode (0x01):** Forces legacy xAPIC mode. • **X2ApicMode (0x02):** Forces x2APIC mode independent of thread count.
PCIe Speed PMM Control	Auto	• **0:** Dynamic link speed determined by power management functionality. • **1:** Static Target Link Speed (Gen4); sets the maximum idle link speed to 16 GT/s. • **Auto/2:** Static Target Link Speed (Gen5); sets the maximum idle link speed to 32 GT/s, thereby disabling the feature).
PCIe ARI Support (SRIOV)	Auto	• **Enabled/Auto:** Enables Alternative Routing ID interpretation. • **Disabled:** Disables Alternative Routing ID interpretation.
PCIe Ten Bit Tag Support	Auto	• **Enabled/Auto:** Enables PCIe 10-bit tags for supported devices. • **Disabled:** Disables PCIe 10-bit tags for all devices.
IOMMU	Auto	• **Enabled/Auto:** Enables IOMMU. AMD recommends setting this to `pt:pass-through` in the Linux kernel settings. • **Disabled:** Disables IOMMU.
AVIC	Disabled	Advanced Virtual Interrupt Controller. • **Disabled:** Disables AVIC. • **Enabled:** Enables AVIC.
x2AVIC	Disabled	x2AVIC is an extension of the advanced virtual interrupt controller. This feature currently requires a custom AMD Linux kernel. • **Disabled:** Disables x2AVIC. • **Enabled:** Enables x2AVIC.

Table 3-5: PCIe, I/O, security, and virtualization BIOS settings

TSME	Auto	• **Auto/Disabled:** Disables transparent secure memory encryption. • **Enabled:** Enables transparent secure memory encryption.
SEV	Disabled	In a multi-tenant environment (such as a cloud), Secure Encrypted Virtualization (SEV) mode isolates virtual machines from each other and from the hypervisor. • **Disabled:** SEV is disabled. • **Enabled:** SEV is enabled.
SEV-ES	Disabled	Secure Encrypted Virtualization-Encrypted State (SEV-ES) mode extends SEV protection to the contents of the CPU registers by encrypting them when a virtual machine stops running. Combining SEV and SEV-ES can reduce the attack surface of a VM by helping protect the confidentiality of data in memory. • **Disabled:** SEV-ES is disabled. • **Enabled:** SEV-ES is enabled.
SEV-SNP	Disabled	Secure Encrypted Virtualization-Secure Nested Paging (SEV-SNP) mode builds on SEV and SEV-ES by adding strong memory integrity protection to create an isolated execution environment that helps prevent malicious hypervisor-based attacks such as data replay and memory re-mapping. SEV-SNP also introduces several additional optional security enhancements that support additional VM use models, offer stronger protection around interrupt behavior, and increase protection against recently-disclosed side channel attacks. • **Disabled:** SEV-SNP is disabled. • **Enabled:** SEV-SNP is enabled.

Table 3-5: PCIe, I/O, security, and virtualization BIOS settings

3.6 Higher-Level Settings

The system powers on to an initial state, after which succeeding software layers may affect system settings:

1. System firmware validates basic hardware functionality and allows users to change various settings via the BIOS Setup menus.
2. UEFI provides a shell environment that allows users to further interact with the system.
3. The operating system or hypervisor is the next software layer that provides control over system hardware.
4. Lastly, certain applications can also affect underlying hardware.

Each of the lines above may alter settings made by prior line, and some user changes require a reboot to take effect.

Please refer to your OEM documentation and/or applicable AMD Tuning Guide(s) for further guidance.

Chapter 4

BIOS Settings

Tuning BIOS settings can improve performance for specific workloads. Evaluate all of the options discussed in this section to determine their impact on your specific workload.

Table 4-1 describes the BIOS options that have the greatest impact to Hadoop Server performance using the BIOS parameters and settings found on AMD Customer Reference Boards (CRB), and OEM settings may vary. Please see your OEM BIOS documentation for platform-specific BIOS information. Please also see the *BIOS & Workload Tuning Guide for AMD EPYC™ 9004 Series Processors* (available from AMD EPYC Tuning Guides) for additional BIOS tuning information.

Name	Value	Description
SMT Control	Enabled	Enables Symmetric Multi-Threading (SMT), which creates two computing threads per core. Consider disabling this setting when using AMD EPYC processors with more than 32 cores.
NUMA Node per Socket (NPS)	NPS1	The default value is 1, but there are times when setting this to 4 boosts performance. This is especially true for MapReduce tasks where only one core is dedicated to each task and for applications that are sensitive to memory latency.
ACPI SRAT L3 Cache as NUMA Domain	Disable	This setting is disabled by default. In some cases, enabling this feature will improve L3 cache hits from the processor, thereby improving memory latency and overall performance. This is especially true when ≤4GB of memory is allocated to a Map or Reduce task.
X3D	Enabled	In an AMD EPYC 9004 processor with AMD 3D V-Cache technology, enabling this option enables the AMD 3D V-Cache module in the CCD to increase the total size of the L3 cache memory to 96MB. Disabling this option reduces the L3 cache in the CCD to 32MB. AMD recommends enabling this option, if available and if doing so benefits your workload after testing. This option is only available on AMD EPYC 9004 Series Processors with AMD 3D V-Cache.

Table 4-1: BIOS Settings for optimal Hadoop performance

This page intentionally left blank.

Chapter 5 Linux Optimizations

Tuning Linux for Hadoop involves looking at how various system subcomponents interact. This is particularly true for network and disk I/O because they often conflict with each other. The goal of this tuning is to optimize each subsystem for throughput while using simple stress test tools to look for and address any performance issues or bottlenecks. This will rule out any performance issues in the subsystems before proceeding to tune the Hadoop instance.

5.1 Network Configuration

The *Linux® Network Tuning Guide for AMD EPYC™ 9004 Series Processors* (available from AMD EPYC Tuning Guides) describes how to properly configure the network for systems in a Hadoop cluster. Per that guide:

- Tune the sizes of the TX and RX rings.
- Change the number of interrupts queues to match the cores on the NUMA node on which the NIC is collocated, and pin those interrupts to the correct CPU cores.
- Use the `iperf` utility to stress test the network infrastructure and verify proper configuration.
- Be sure to follow OS IOMMU setting recommendations, because this will significantly impact system performance. This normally entails setting the IOMMU to passthrough mode by adding the kernel parameter `iommu=pt` on the kernel boot line. To do this, modify `/etc/default/grub` and then run either:
  - **SLES or RHEL:** `grub2-mkconfig`
  - **Ubuntu:** `sudo update-grub`
  - Setup Reverse DNS to ensure that a node's hostname can be looked up via its IP address. Both Kerberos authentication and certain Hadoop functionalities utilize and require reverse DNS.

5.2 Disk Configuration

Look at the following parameters when optimizing disk I/O in a Hadoop environment:

- Partition alignment. This is more critical when using SSD and NVMe drives because misaligning partitions will increase disk I/O operations and significantly impact performance.
- Scheduler
- Queue depth
- Number of requests allowed
- Filesystem mount options.

5.2.1 Storage Device to CPU Core Ratios

The ratios in Table 5-1 are guidelines to help you determine the minimum number of devices needed to establish ideal I/O throughput.

Drive Type	Drive to Core Ratio
10K rpm SATA Hard Drives	1 drive for every core
SATA Solid State Disks	1 drive for every 2 to 4 cores
PCIe Generation 3 NVMe Drive	1 drive for every 4 to 8 cores
PCIe Generation 4 NVMe Drive	1 drive for every 8 to 16 cores
PCIe Generation 5 NVMe Drive	1 drive for every 16 to 20 cores

Table 5-1: Storage Device Type to CPU Core Ratio

The best scheduler option for most Hadoop environments is either `deadline` or `noop`. There is no standard, so test both options in your environment. You can make system-wide scheduler changes via the Linux boot command line using the `elevator=<scheduler>` kernel option. You can view and modify a device scheduler post-boot using `/sys/block/<device>/queue/scheduler`. For example, if the square brackets (`[` and `]`) are around `deadline`, then the device is configured to use the `deadline` scheduler. You can execute the command `echo noop` into the file for `device sda` and then list the contents of the file again to see that the square brackets are now around the `noop` scheduler.

```
cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
echo noop > /sys/block/sda/queue/scheduler
cat /sys/block/sda/queue/scheduler
[noop] deadline cfg
```

Set `/sys/block/<device>/device/Queue_depth` and `/sys/block/<device>/queue nr_requests` to ensure they don't overwhelm the device. Use performance monitoring tools such as `sar` or `iostat` to look for large queue times and adjust these settings as needed to optimize performance.

Formatting and tuning options are important. Tools like `tunefs` let you set the reserve size to zero percent, which can free over 25 GB on a 1 TB disk. If the underlying file system contains many large files, then you can free more space by lowering the number of `inodes` at format time accordingly.

Most Hadoop deployments use either the ext4 or xfs filesystem, while HPE Ezmeral Data Fabric (formerly MapR™) deploys on unformatted disks. The `noatime` filesystem option is important for any Hadoop data mounts. Include the `discard` option if your system is running on an SSD that supports this feature. You can set these options using the `-o` parameter when manually executing the mount command, or include them in the options section of the data drive entries in `fstab`, such as by either:

- Executing the command `mount -o noatime /dev/sda1 /data/1`
- Modifying `/etc/fstab` for filesystems that are mounted during the boot process. For example:

```
UUID=a6f3ee72-5fef-4ec5-8074-49f4ae109a88 /data/1 xfs defaults,noatime 0 0
UUID=30375f23-b704-4b3b-ae33-349bb0dac416 /data/2 xfs defaults,noatime 0 0
UUID=0c347985-6c1f-469e-918e-ae6ad4ee5002 /data/3 xfs defaults,noatime 0 0
UUID=a970c80a-c9ea-47c4-9fbe-acddf9ab7ffb /data/4 xfs defaults,noatime 0 0
UUID=69045795-a472-477c-9115-35bc4b940224 /data/5 xfs defaults,noatime 0 0
UUID=3531c522-3f82-4b9e-876f-ee44e133dc3c /data/6 xfs defaults,noatime 0 0
UUID=fa96cd17-37a0-441b-b001-527e7765de42 /data/7 xfs defaults,noatime 0 0
UUID=5036062a-06a2-4227-98b3-a1224cb3f649 /data/8 xfs defaults,noatime 0 0
```

Use the `fio` test to stress test the disk subsystems and determine maximum disk throughput in both I/O per second (IOPS) and MB/sec. This utility can also see whether any disks are running slower than the rest.

The `java.io.FileNotFoundException` (too many open files) error will occur, and jobs will fail if the default number of open files is too small. To avoid this, execute the `ulimit` command as the root user to increase the number of open files to 32768 or even 65536, such as `ulimit -n 65536`. Add the following lines to `/etc/security/limits.conf`:

```
*      -      nofile     65536
*      -      nproc      65536
```

You may sometimes need to increase the system-wide `fs.file-max` setting. To do this, either:

- Execute the command `sysctl -w fs.file-max <maximum number of open files>`
- Make this setting persistent across reboots by adding `fs.file-max=<maximum number of open files>` to `/etc/sysctl.conf`

5.3 Memory Configuration

Use all memory channels evenly when populating memory. To do this:

- Place the same number and size memory DIMMs in each memory channel to keep the NUMA nodes balanced.
- Either:
  - Use the OS `sysctl` utility to set `vm.overcommit_memory` and `vm.swappiness` to minimize swapping, and to set `vm.dirty_ratio` and `vm.dirty_background_ratio` to tune down the OS cache buffers. For example:

```
sysctl -w vm.overcommit_memory=0 vm.swappiness=1 vm.dirty_ratio=20
vm.dirty_background_ratio=1
```

  - Make these settings persistent across reboots by adding them to `/etc/sysctl.conf`, as described in "Example SLES 12 Server Configuration Files" on page 26.
  - Disable Transparent Huge Pages.

5.4 CPU Configuration

Each compute node should ideally have 32 processor cores because the YARN scheduler has some inefficiencies for both MapReduce and Spark frameworks at core counts above ~48 node.

Check the performance governor on all processor cores. SLES allows you to:

- View the available frequency governors by executing the `cpupower frequency-info -governor` command.
- Set the frequency governor by executing the `cpupower frequency-set -governor performance` command.

The `tuned-adm throughput-performance` profile sets the governor to performance, which generally works best for Hadoop workloads.

5.4.1 Potential Alternative for Higher Core Count Platforms

Implementing containers or virtualization allows users to use Hadoop nodes with higher core counts. Either of these options allow you to carve out a container or VM that is 32 cores or less while leveraging AMD EPYC memory and I/O resources. Hadoop allows this kind of consolidation, but software licensing costs could increase. Consult your Hadoop software vendor licensing agreement to ensure optimal TCO. When running in virtual environments, binding the VM to specific cores and memory within a NUMA node or LLC is usually going to improve performance.

5.4.2 Example SLES 12 Server Configuration Files

This sample configuration file includes the tuning options described in the preceding sections of this chapter.

```
/etc/default/grub
GRUB_CMDLINE_LINUX="mitigations=auto splash=silent quiet showopts crashkernel=207M,high
crashkernel=72M,low iommu=pt"

/etc/init.d/boot.local
#configure Mellanox network card.
mlnx_tune -p HIGH_THROUGHPUT
set_irq_affinity_bynode.sh 3 ens2f0 ens2f1
#disable transparent hugepages
echo never > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/enabled
#disk tuning
for a in sda sdb sdc sdd sde sdf sdg sdh
do
  echo “deadline” > /sys/block/${a}/queue/scheduler
  echo 512 > /sys/block/${a}/queue/nr_requests
done

/etc/sysctl.conf
#Network Tunings
net.ipv4.conf.default.rp_filter=1
net.ipv4.tcp_timestamps=0
net.ipv4.tcp_sack = 1
net.core.netdev_max_backlog = 25000
net.core.rmem_max = 2147483647
net.core.wmem_max = 2147483647
net.core.rmem_default = 33554431
net.core.wmem_default = 33554432
net.core.optmem_max = 33554432
net.ipv4.tcp_rmem =8192 33554432 2147483647
net.ipv4.tcp_wmem =8192 33554432 2147483647
net.ipv4.tcp_low_latency=1
net.ipv4.tcp_adv_win_scale=1
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv4.conf.all.arp_filter=1
net.ipv4.tcp_retries2=5
net.ipv6.conf.lo.disable_ipv6 = 1
net.core.somaxconn = 65535
#memory cache settings
vm.swappiness=1
vm.overcommit_memory=0
vm.dirty_background_ratio=1
vm.dirty_ratio=20
```

Chapter 6 Hadoop Settings

The Hadoop Distributed File System (HDFS) and Yet Another Resource Negotiator (YARN) are the main components that form the data management layer for Hadoop, where:

- HDFS is the storage management framework.
- YARN is the resource management framework.

There are no universal settings for all Hadoop environments. The settings you use are highly dependent on workload characteristics and hardware subsystem choices. This chapter provides brief descriptions and tuning guidelines for various interactive Hadoop.

6.1 YARN Settings

YARN includes two major component daemons:

- **Resource Manager:** See "Resource Manager" on page 27.
- **Node Manager:** See "Node Manager" on page 28.

6.1.1 Resource Manager

ResourceManager is the master YARN daemon. It manages resource assignment among the competing applications. The scheduler is most significant part of the ResourceManager because most Hadoop installations rely on either:

- The Fair scheduler, which tries to assign resources equally across jobs over time.
- The Capacity scheduler, which tries to give each user or job a minimum capacity guarantee while allowing any excess capacity not utilized by one user or organization to be accessed by others.

Note: If only one job is running at a time, then either scheduler will work.

The MapReduce concept often breaks a job into hundreds or even thousands of tasks and reduces the task footprint to a single thread and smaller memory footprint. This footprint represents the minimum container size allocations for both memory and vCPU cores. As a general rule of thumb, calculate this value by dividing the total memory available to the YARN NodeManager (`yarn.nodemanager.resource.memory-mb`) by the number of vCPU cores (`yarn.nodemanager.resource.vcores`) available to it. In other cases, large container applications such as. SPARK or Hive with LLAP can have large container memory footprints and many vCPU cores. These latter cases require considering the AMD EPYC processor's NUMA architecture, specifically the number of CPU cores and the amount of memory available per NUMA node. Tune these settings in either:

- The cluster management software (Cloudera Manager for CDH, Ambari for HDP, HPE Ezmeral Data Fabric Control System for HPE Ezmeral Data Fabric).
- Directly in the `yarn-site.xml` files in a manual Apache Hadoop configuration.

6.1.1.1 Settings for Resource Consumption Control

The following are the scheduler settings for controlling resource consumption for individual YARN containers. These settings define the minimum, maximum, and stepping allocation of memory and vcore (virtual CPU) resources of the YARN containers in the cluster.

YARN Scheduler Settings and Explanations		
Name	**Value**	**Description**
yarn.scheduler.minimum-allocation-vcores	1	Minimum vCPU cores allocated per YARN container.
yarn.scheduler.maximum-allocation-vcores	Restrict to vCPU cores in a NUMA node or LLC if **LLC as NUMA** enabled	Maximum vCPU cores allocated per YARN container.
yarn.scheduler.increment-allocation-vcores	1	Incremental step from min to max vCPU core allocation per YARN container.
yarn.scheduler.minimum-allocation-mb	Yarn.nodeman-ager.resource.memory-mb / yarn.nodemanager.resource-cpu-vcpus	Minimum memory to allocate to each YARN container.
yarn.scheduler.maximum-allocation-mb	Application dependent. Keep this less than the total memory in a NUMA node or LLC, if **LLC as NUMA** enabled.	Maximum memory allocated to a YARN container.
yarn.scheduler.increment-allocation-mb	512 or 1024	Incremental step from min to max memory allocation per container.

Table 6-1: YARN Scheduler settings

6.1.2 Node Manager

The NodeManager runs on each of the worker nodes. The key settings for proper NodeManager configuration are specifying the maximum processing threads and memory Hadoop can use on a particular worker node. These properties are what control how much of a node's resources YARN can use.

YARN NodeManager Settings and Explanations	
Name	**Description**
yarn.nodemanager.resource.memory-mb	Total memory in worker node to be used by NodeManager.
Yarn.nodemanager.resource.cpu-vcores	Total vCPU cores in the worker node to be used by NodeManager.

Table 6-2: YARN NodeManager settings

Different Hadoop vendors offer recommended settings with worksheets and scripts for some of these based upon any other Hadoop components installed on your nodes (e.g. HIVE, Spark, etc.). For example, configure a datanode with a single 32-core/64-thread AMD EPYC 9004 Series Processor and 512 GB of memory as follows:

```
yarn.scheduler.minimum-allocation-vcores: 1
yarn.scheduler.maximum-allocation-vcores: 4
yarn.scheduler.increment-allocation-vcores: 1
yarn.scheduler.minimum-allocation-mb: 1024
yarn.scheduler.maximum-allocation-mb: 32768
yarn.scheduler.increment-allocation-mb: 1024
yarn.nodemanager.resource.memory-mb: 507904
yarn.nodemanager.resource.cpu-vcores: 64
yarn.nodemanager.local-dirs: <List of local directories (one per disk) that YARN will use
to store intermediate and temporary data.>
```

Using native task libraries on map outputs can significantly boost performance. Native libraries include:

- Components for compression codecs (`bzip2`, `gzip`, `lzo` `lz4`, `snappy`, and `zlib`).
- Native I/O utilities for HDFS Short-Circuit Local Reads and Centralized Cache Management.
- CRC32 checksum implementation

High I/O (shuffle) intensive applications can see significant performance gains from using these libraries. CDH 5.14 includes the **Enable Optimized Map-side Output Collector** checkbox to enable this feature.

Verify that the native libraries are properly installed by executing the `hadoop checknative -a` command:

```
# hadoop checknative -a
21/02/23 05:55:33 INFO bzip2.Bzip2Factory: Successfully loaded & initialized native-bzip2
library system-native
21/02/23 05:55:33 INFO zlib.ZlibFactory: Successfully loaded & initialized native-zlib
library
Native library checking:
hadoop:  true /opt/cloudera/parcels/CDH-7.1.4-1.cdh7.1.4.p0.6300266/lib/hadoop/lib/native/
libhadoop.so.1.0.0
zlib:    true /lib64/libz.so.1
zstd  :  true /opt/cloudera/parcels/CDH-7.1.4-1.cdh7.1.4.p0.6300266/lib/hadoop/lib/native/
libzstd.so.1
snappy:  true /opt/cloudera/parcels/CDH-7.1.4-1.cdh7.1.4.p0.6300266/lib/hadoop/lib/native/
libsnappy.so.1
lz4:     true revision:10301
bzip2:   true /usr/lib64/libbz2.so.1
openssl: true /usr/lib64/libcrypto.so
ISA-L:   true /lib64/libisal.so.2
```

AMD also recommends checking the map task system logs when expected native task utilization is enabled and looking for the following message(s):

```
2021-02-23 10:42:45,474 INFO [main] org.apache.hadoop.mapred.nativetask.NativeRuntime:
Nativetask JNI library loaded.
2021-02-23 10:42:45,523 INFO [main]
org.apache.hadoop.mapred.nativetask.NativeBatchProcessor: NativeHandler: direct buffer
size: 1048576
2021-02-23 10:42:45,545 INFO [main] org.apache.hadoop.mapred.nativetask.util.OutputUtil:
nativetask.output.manager = org.apache.hadoop.mapred.nativetask.util.NativeTaskOutputFiles
2021-02-23 10:42:45,546 INFO [main]
org.apache.hadoop.mapred.nativetask.NativeMapOutputCollectorDelegator: Native output
collector can be successfully enabled!
2021-02-23 10:42:45,546 INFO [main] org.apache.hadoop.mapred.MapTask: Map output collector
class = org.apache.hadoop.mapred.nativetask.NativeMapOutputCollectorDelegator2018-08-23
15:30:36,019 INFO [main] org.apache.hadoop.mapred.nativetask.NativeRuntime: Nativetask JNI
library loaded.
```

6.2 HDFS Settings

This section describes the main parameters that will need testing with your workload. As before, you can make these changes via either the cluster management software (Cloudera Manager for CDH or Ambari for HDP) or by modifying `hdfs-site.xml` directly.

The HDFS (Hadoop Distributed File System) is the heart of the Hadoop platform. Regardless of which other projects/ applications you use (Hive, Spark, Kafka etc), they all depend on the HDFS filesystem. As such, HDFS performance is undeniably critical to the overall platform availability, performance, and throughput.

Note: HPE Ezmeral Data Fabric has its own filesystem that is managed via the HPE Ezmeral Data Fabric Control System.

Various components like Hive, Spark, HBase, Kafka etc use HDFS as their filesystem. As such, the HDFS acts as a single point of failure for the data platform. This means that availability, performance, and throughput are largely governed by that of HDFS performance and availability.

HDFS has many configurations, many of which depend on the specific use of the data platform. Keeping those aside, here are some features that you must consider enabling on the platform:

- Enable High Availability with Automatic Failover*.
- Optimize RPC performance using:
  - **Service RPC:** The service RPC port gives the DataNodes a dedicated port to report their status via block reports and heartbeats. The port is also used by Zookeeper Failover Controllers for periodic health checks by the automatic failover logic. The port is never used by client applications and hence reduces RPC queue contention between client requests and DataNode messages.
  - **Tuning handler counts:** `dfs.datanode.handler.count`, `dfs.namenode.handler.count`, and `dfs.namenode.service.handler.count`. Generally, `dfs.namenode.service.handler.count` = 50% of `dfs.namenode.handler.count`.
  - **RPC Congestion Control:** This feature was designed to help Hadoop services respond more predictably under high loads. RPC queue overflow can cause request timeouts and eventual job failures. This problem can be mitigated if the NameNode sends an explicit signal back to the client when its RPC queue is full. Instead of waiting for a request that may never complete, the client throttles itself by re-submitting the request with an

exponentially increasing delay. If you are familiar with the Transmission Control Protocol, this is similar to how senders react when they detect network congestion.

You can enable this feature using the following setting in `core-site.xml`, being sure to replace 8020 with your NameNode RPC port number if it is different. Do not enable this setting for the Service RPC port or the DataNode lifeline port.

```
ipc.8020.backoff.enable=true
```

- **RPC Faircall Queue:** This replaces the single RPC queue with multiple prioritized queues (not to be confused with Yarn queues). The RPC server maintains a history of recent requests grouped by user. It places incoming requests into an appropriate queue based on the user's history. RPC handler threads will dequeue requests from higher priority queues with a higher probability. `FairCallQueue` complements RPC congestion control very well and works best when you enable both features together. `FairCallQueue` can be enabled with the following `core-site.xml` setting. Replace 8020 with your NameNode RPC port if it is different. Do not enable this setting for the Service RPC port or the DataNode lifeline port.

  ```
  ipc.8020.callqueue.impl= org.apache.hadoop.ipc.FairCallQueue
  ```

- **Datanode Lifeline Protocol:** Datanode Lifeline Protocol introduces a new lightweight RPC message that is used by the DataNodes to report their health to the NameNode. It was developed in response to problems seen in some overloaded clusters where the NameNode was too busy to process heartbeats and spuriously marked DataNodes as dead.

- **Tune the Datanode and Namenode Heap memory size:** For the Datanode, this is the number of replicas of its hosts. For the Namenode, this is the number of files on the system govern the heap size requirements. A conservative approach is to set a heap size of 1 GB for every million blocks stored in the HDFS.

- Optimal logging:

  - Enable async logging in `hdfs-site.xml`
    ```
    dfs.namenode.edits.asynclogging = true
    dfs.namenode.audit.log.async = true
    ```

  - Minimize StateChange logging (`log4j.properties`):
    ```
    log4j.logger.BlockStateChange=WARN
    log4j.logger.org.apache.hadoop.hdfs.StateChange=WARN
    ```

- **dfs.datanode.data.dir:** List of directories where HDFS data will be stored (one per disk).

- **dfs.blocksize:** Default HDFS block size. Jobs can overwrite this setting.

6.2.1 Erasure Coding

Erasure coding attempts to achieve the same or better protection against data loss as three-way replication while reducing the total amount of stored data. Using the RS-6-3-1024K erasure coding policy breaks HDFS files down into block groups with each block group consisting of 6 blocks of data and 3 blocks of parity. This creates a 50% overhead for redundant storage and thus requires 150% of the total data stored. By contrast, replication-3 makes 3 copies of the data and thus requires 300% of the total data stored. Erasure coding thus offers the same level of protection as replication-3 while needing only half the disk space.

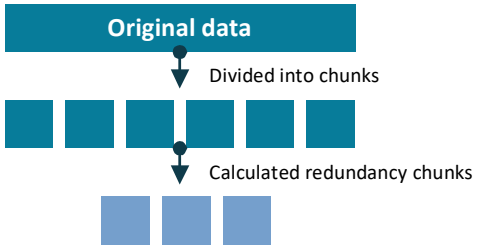

Figure 6-1: Data chunking

Each block in a block group is stored on a different DataNode, and at least one DataNode is therefore required to obtain the full benefit of erasure coding. Erasure coding places additional compute and network stress on the data nodes, making it important to consider the balance between storage gains and resource consumption.

Erasure Coding has been around for a while but is relatively new to HDFS. This feature can achieve significant performance and storage benefits, and AMD thus recommends testing it in development before attempting to use it in any production environment.

6.3 NUMA Optimization for Dual-socket Servers

NUMA is a computer memory design used in multiprocessing where the memory access time depends on the memory location relative to the processor. NUMA allows a processor to access its local memory faster than non-local memory, that is, memory that is either local to another processor or memory that is shared between processors. Hadoop YARN containers can benefit from NUMA design and achieve higher performance by binding a specific NUMA node and all subsequent memory allocations served by the same node to reduce remote memory accesses. Also, the NUMA nodes allocated for YARN containers within sockets have better performance compared to NUMA nodes allocated across the sockets.

6.3.1 NUMA Setting

To modify the NUMA configuration:

1. Configure the NodeManager to use the LinuxContainerExecutor (LCE) in `yarn-site.xml`:

```
yarn.nodemanager.container-executor.class =
org.apache.hadoop.yarn.server.nodemanager.LinuxContainerExecutor
yarn.nodemanager.linux-container-executor.group = Hadoop
```

2. Configure the `container-executor.cfg` file required by the LCE and read by the container-executor program:

```
yarn.nodemanager.linux-container-executor.group=hadoop
banned.users=hdfs,yarn,mapred,bin
min.user.id=1000
feature.terminal.enabled=1
```

3. Enable the NodeManager NUMA awareness feature for the containers.

```
yarn.nodemanager.numa-awareness.enabled = true
```

4. Enable the NUMA topology reading, which decides whether to read the NUMA topology from the system or from the configurations. Setting this property to `true` reads the topology system by executing the `numactl -hardware` command in UNIX systems, and similarly in Windows.

```
yarn.nodemanager.numa-awareness.read-topology = true
```

6.3.2 NUMA Source Code Tuning

This section describes some optimizations that AMD has been looking at to improve CPU and memory allocation. NUMA source code tuning can significantly improve performance, especially on dual-socket systems. This requires you to modify Hadoop source code and may violate support agreements. Check with your Hadoop vendor/service provider to verify that this is allowed before attempting these changes in production environments.

Hadoop allocates node CPU and memory resources to allow containers to execute tasks. This may occur across both sockets of a dual-socket AMD EPYC-powered server depending on the existing Hadoop NUMA source code. This can cause long-latency memory access between the two sockets and degrade performance accordingly. Consider modifying the Hadoop NUMA source code to allocate CPU and memory resources within one socket to remove cross-socket memory access and improve performance. The modified algorithm and Hadoop source code appear below.

The Hadoop NUMA source code uses the `numactl` command to bind memory and CPU resources to a task. You can use `membind` as the argument in the `numactl` command in the Hadoop source code, which enables local memory allocations from the nodes.To do this, modify the `public List<PrivilegedOperation> preStart(Container container)` function in `NumaResourceHandlerImpl.java` by changing the `interleave` line to `membind`:

```
"--membind=" + String.join(",", numaAllocation.getMemNodes()))
//"--interleave=" + String.join(",", numaAllocation.getMemNodes())
```

To modify the algorithm for Hadoop resource allocation:

1. Search every NUMA node in the dual-socket server to verify that the node can provide enough resources for the Hadoop container. Return this node if the node can provide enough resources, else proceed Step to 2.

2. Check NUMA nodes for the memory resources on Socket 1 and Socket 2 separately.

3. Check NUMA nodes for the CPU resources on Socket 1 and Socket 2 separately.

4. If the NUMA nodes in Socket 1 have enough memory and CPU resources for the Hadoop container, then allocate these nodes in Socket 1. Similarly, if the NUMA nodes in Socket 2 have enough memory and CPU resources for the Hadoop container, then allocate these nodes to Socket 2.

The following example shows this algorithm implementation. In `NumaResourceAllocator.java` file, the function `private NumaResourceAllocation allocate(ContainerId containerId, Resource resource)` is modified as shown below:

```
 private NumaResourceAllocation allocate(ContainerId containerId,
      Resource resource) {
// check every NUMA node for the Hadoop container resource requirement
 for (int index = 0; index < numaNodesList.size(); index++) {
      NumaNodeResource numaNode = numaNodesList.get(index);
```

```
      if (numaNode.isResourcesAvailable(resource)) {
        numaNode.assignResources(resource, containerId);
        LOG.info("Assigning NUMA node " + numaNode.getNodeId() + " for memory, "
            + numaNode.getNodeId() + " for cpus for the " + containerId);
        return new NumaResourceAllocation(numaNode.getNodeId(),
            resource.getMemorySize(), numaNode.getNodeId(),
            resource.getVirtualCores());
      }
    }

    // If there is no single node matched for the container resource
    // Check the NUMA nodes for Memory resources from socket 1 and socket 2 respectively.
    long memoryRequirement = resource.getMemorySize();
Map<String, Long> memoryAllocations = Maps.newHashMap();
for (int i = 0; i < numaNodesList.size() / 2; ++i) {
      long memoryRemaining = numaNodesList.get(i).
          assignAvailableMemory(memoryRequirement, containerId);
      memoryAllocations.put(numaNodesList.get(i).getNodeId(),
          memoryRequirement - memoryRemaining);
      memoryRequirement = memoryRemaining;
      if (memoryRequirement == 0) {
        break;
      }
    }

    long memoryRequirement1 = resource.getMemorySize();
    Map<String, Long> memoryAllocations1 = Maps.newHashMap();
      for (int i = numaNodesList.size() / 2; i < numaNodesList.size(); ++i) {
      long memoryRemaining = numaNodesList.get(i).
          assignAvailableMemory(memoryRequirement1, containerId);
      memoryAllocations1.put(numaNodesList.get(i).getNodeId(),
          memoryRequirement1 - memoryRemaining);
      memoryRequirement1 = memoryRemaining;
      if (memoryRequirement1 == 0) {
        break;
      }
    }

    if (memoryRequirement != 0 || memoryRequirement1 != 0) {
      LOG.info("There is no available memory:" + resource.getMemorySize()
          + " in numa nodes for " + containerId);
      releaseNumaResource(containerId);
      return null;
    }

    // Check the NUMA nodes for CPU resources from socket 1 and socket 2 respectively.
    int cpusRequirement = resource.getVirtualCores();
    Map<String, Integer> cpuAllocations = Maps.newHashMap();
    for (int index = 0; index < numaNodesList.size() / 2; index++) {
      int cpusRemaining = numaNodesList.get(index).
          assignAvailableCpus(cpusRequirement, containerId);
      cpuAllocations.put(numaNodesList.get(index).getNodeId(), cpusRequirement -
cpusRemaining);
      cpusRequirement = cpusRemaining;
      if (cpusRequirement == 0) {
        break;
      }
    }

    int cpusRequirement1 = resource.getVirtualCores();
    Map<String, Integer> cpuAllocations1 = Maps.newHashMap();
    for (int index = numaNodesList.size() / 2; index < numaNodesList.size(); index++) {
      int cpusRemaining = numaNodesList.get(index).
          assignAvailableCpus(cpusRequirement1, containerId);
```

```
      cpuAllocations1.put(numaNodesList.get(index).getNodeId(), cpusRequirement1 -
cpusRemaining);
      cpusRequirement1 = cpusRemaining;
      if (cpusRequirement1 == 0) {
        break;
      }
    }

// allocate nodes from socket 1 if there are enough CPU and memory for the Hadoop
containers
    if (memoryRequirement == 0 && cpusRequirement == 0) {
    NumaResourceAllocation assignedNumaNodeInfo =
        new NumaResourceAllocation(memoryAllocations, cpuAllocations);
    LOG.info("Assigning multiple NUMA nodes ("
        + StringUtils.join(",", assignedNumaNodeInfo.getMemNodes())
        + ") for memory, ("
        + StringUtils.join(",", assignedNumaNodeInfo.getCpuNodes())
        + ") for cpus for " + containerId);
        return assignedNumaNodeInfo;
        } else if ( memoryRequirement1 == 0 && cpusRequirement1 == 0) {
  // allocate nodes from socket 2 if there are enough CPU and memory for the Hadoop
containers
    NumaResourceAllocation assignedNumaNodeInfo =
        new NumaResourceAllocation(memoryAllocations1, cpuAllocations1);
    LOG.info("Assigning multiple NUMA nodes ("
        + StringUtils.join(",", assignedNumaNodeInfo.getMemNodes())
        + ") for memory, ("
        + StringUtils.join(",", assignedNumaNodeInfo.getCpuNodes())
        + ") for cpus for " + containerId);

      return assignedNumaNodeInfo;
    }
    releaseNumaResource(containerId);
    return null;
}
```

This page intentionally left blank.

Chapter 7 Resources

- Memory Population Guidelines for AMD EPYC 9004 Series Processors (login required)
- Socket SP5 Platform NUMA Topology for AMD Family 19h Models 10h–1Fh (login required)
- From AMD EPYC Tuning Guides:
  - *Linux® Network Tuning Guide for AMD EPYC™ 9004 Series Processors*
  - *Windows® Network Tuning Guide for AMD EPYC™ 9004 Series Processors*
- Tuning YARN*

This page intentionally left blank.

Chapter 8

Processor Identification

Figure 8-1 shows the processor naming convention for AMD EPYC 9004 Series Processors and how to use this convention to identify particular processors models:

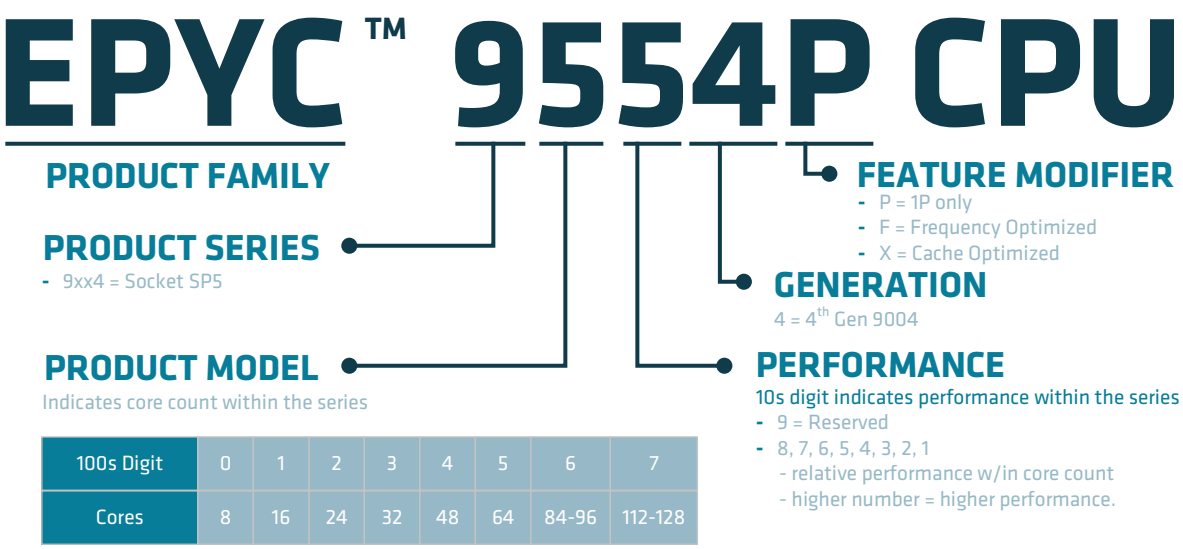

Figure 8-1: AMD EPYC SoC naming convention

8.1 CPUID Instruction

Software uses the `CPUID` instruction (`Fn0000_0001_EAX`) to identify the processor and will return the following values:

- **Family:** 19h identifies the "Zen 4" architecture
- **Model:** Varies with product. For example, EPYC Family 19h, Model 10h corresponds to an "A" part "Zen 4" CPU.
  - **91xx-96xx (including "X" OPNs):** Family 19h, Model 10-1F
  - **97xx:** Family 19h, Model A0-AF
- **Stepping:** May be used to further identify minor design changes

For example, `CPUID` values for Family, Model, and Stepping (decimal) of 25, 17, 1 correspond to a "B1" part "Zen 4" CPU.

8.2 New Software-Visible Features

AMD EPYC 9004 Series Processors introduce several new features that enhance performance, ISA updates, provide additional security features, and improve system reliability and availability. Some of the new features include:

- 5-level Paging
- AVX-512 instructions on a 256-byte datapath, including BFLOAT16 and VNNI support.
- Fast Short Rep STOSB and Rep CMPSB

Not all operating systems or hypervisors support all features. Please refer to your OS or hypervisor documentation for specific releases to identify support for these features.

Please also see the latest version of the *AMD64 Architecture Programmer's Manuals* or *Processor Programming Reference (PPR) for AMD Family 19h*.

8.2.1 AVX-512

AVX-512 is a set of individual instructions supporting 512-bit register-width data (i.e., single instruction, multiple data [SIMD]) operations. AMD EPYC 9004 Series Processors implement AVX 512 by "double-pumping" 256-bit-wide registers. AMD's AVX-512 design uses the same 256-bit data path that exists throughout the Zen4 core and enables the two parts to execute on sequential clock cycles. This means that running AVX-512 instructions on AMD EPYC 9004 Series will cause neither drops on effective frequencies nor increased power consumption. On the contrary, many workloads run more energy-efficiently on AVX-512 than on AVX-256P.

Other AVX-512 support includes:

- Vectorized Neural Network Instruction (VNNI) instructions that are used in deep learning models and accelerate neural network inferences by providing hardware support for convolution operations.
- Brain Floating Point 16-bit (BFLOAT16) numeric format. This format is used in Machine Learning applications that require high performance but must also conserve memory and bandwidth. BFLOAT16 support doubles the number of SIMD operands over 32-bit single precision FP, allowing twice the amount of data to be processed using the same memory bandwidth. BFLOAT16 values mantissa dynamic range at the expense of one radix point.