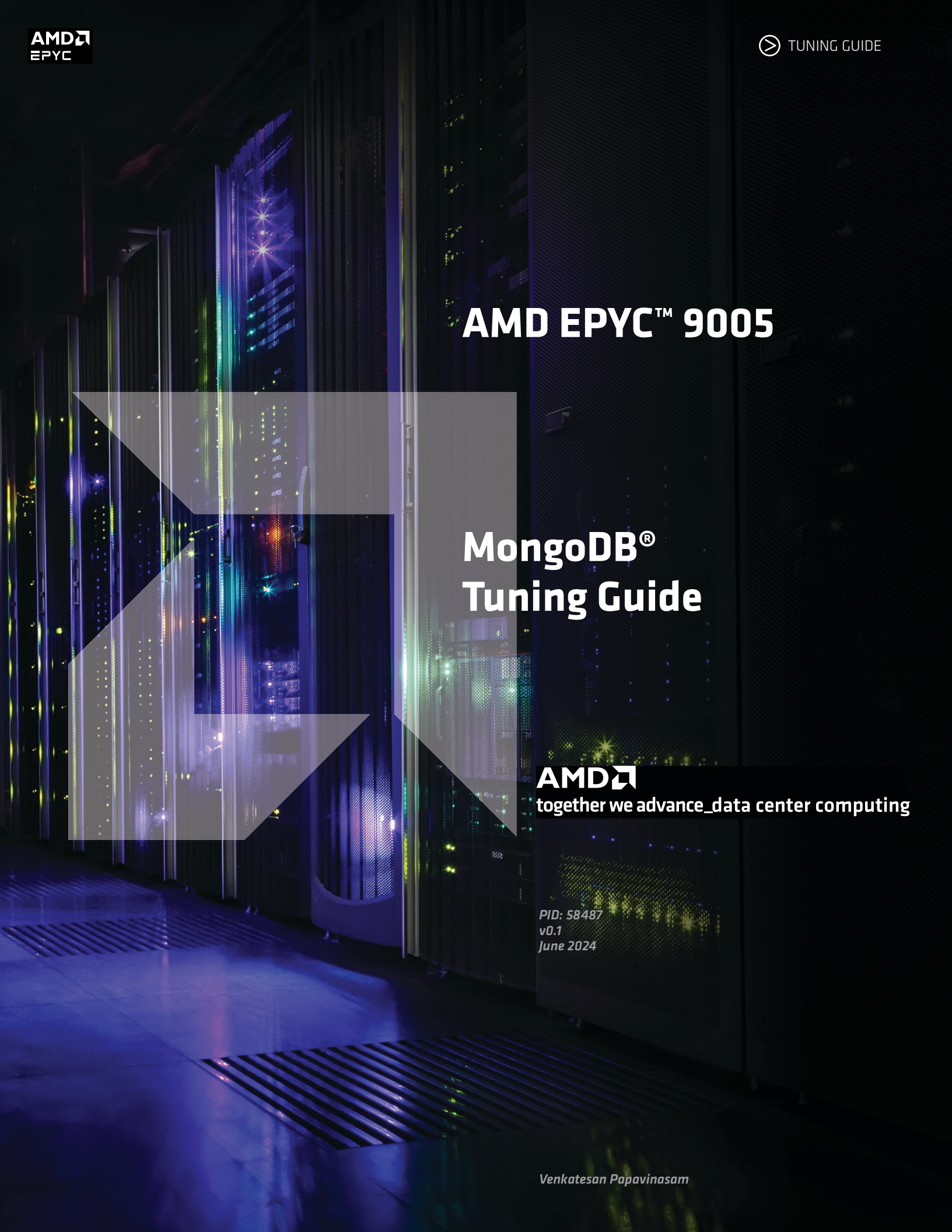

AMD EPYC

TUNING GUIDE

AMD EPYC™ 9005

MongoDB®
Tuning Guide

AMD
together we advance_data center computing

PID: 58487
v0.1
June 2024

Venkatesan Papavinasam

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

AUDIENCE

This document describes best practices for optimizing performance when using MongoDB®. It is intended for a technical audience such as MongoDB 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.

MONGODB®
TUNING GUIDE

CONTENTS

Chapter 1: Introduction

This tuning guide provides detailed descriptions of parameters that can optimize performance on servers built with AMD EPYC™ 9005 Series processors. Default OEM hardware and BIOS settings may require additional tuning to obtain optimal performance on all OS platforms and for all workloads. This guide helps you tune the following settings to optimize the performance of your specific workload:

- Hardware configuration
- BIOS
- Network settings
- OS kernel parameters
- MongoDB® performance tuning

1.1 - 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*

1.2 - MongoDB®

MongoDB® is a cross-platform document-oriented database developed by MongoDB Inc. and licensed under the Server-Side Public License. It is classified as a NoSQL database and uses JSON-like documents with optional schemas.

MongoDB is built around an intelligent distributed systems architecture that allows developers to place data where apps and users need it. MongoDB can run within and across geographically distributed data centers and cloud regions to provide high availability, workload isolation, scalability, and data locality.

1.2.1 - When to Use MongoDB

Brewer's theorem (more commonly known as the CAP theorem) says that a distributed computer system can simultaneously guarantee two but not all three of the following:

- **Consistency:** All nodes see the same data at the same time.
- **Availability:** Every request is guaranteed to receive a response about whether it succeeded or failed.
- **Partition Tolerance:** The system continues operating despite arbitrary message loss.

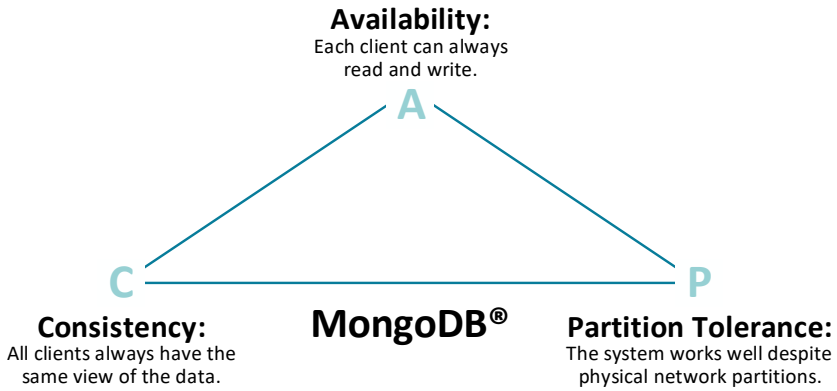

Figure 1-1: MongoDB distributed system

MongoDB is built with a scale-out architecture that allows many small machines to work together to create fast systems that handle huge amounts of data. MongoDB sits at the bottom of the CAP triangle satisfying Consistency and Partition Tolerance. Document databases are extremely flexible because they support varying document structures and can store partially-complete documents. One document can even have others embedded in it.

MongoDB uses JSON-like documents that can have varied structures and the MongoDB query language to allow access to the stored data. Being schema-free means you can create documents without having to first create the structure for the document.

MongoDB allows a significant degree of freedom to design a schema. The first step in optimizing performance is understanding your application's query patterns so that you can design the data model and select the appropriate indexes. It can therefore be a good choice in situations where you have no clear schema definition but need scalability and caching. Sample use cases include real time analytics for mobile apps, content management, and IoT applications.

1.2.2 - Recommended CPUs for MongoDB Servers

Table 1-1 includes basic AMD EPYC processor selection guidelines for general MongoDB use cases. Select the processor that best fits your needs.

System Type	Cores	CPUs[1]	Memory	Storage	Network	Min # of Shards	Min. Node #[2]
Micro	8	1 x AMD EPYC™ 8-core	64 GB	1-4 SAS SSDs	1-5 Gbit	1 shard with 3 replica sets	1+2=3
Small	16	1 x AMD EPYC™ 16-core	128 GB	1-4 SAS SSDs	10 Gbit	1 shard with 3 replica sets	3+2+1=6
Medium	24	1 x AMD EPYC™ 24-core	128 GB / 256 GB	1-4 SAS SSDs / NVMe SSDs	25 Gbit	1 shard with 3 replica sets	3+2+1=6
Large	32	1 x AMD EPYC™ 32-core	256 GB / 512 GB	1-4 NVMe SSDs	25 Gbit	1 shard with 3 replica sets	3+2+1=6
Cloud[3]: PaaS, IaaS	Please see https://www.mongodb.com/docs/atlas/cluster-config/multi-cloud-distribution/* for additional information.						
1 - Please see AMD EPYC Processors for more information. 2 - Total number of nodes = Replica set + Config server + Mongos 3 - Recommendations for Cloud Service Providers (CSP) / Private Cloud for Hypervisor configuration.							

Table 1-1: Recommended AMD EPYC processors for MongoDB servers

AMD recommends using 8, 16, or 32 vCPU instances with 1:4 vCPU to memory ratio VMs with appropriate storage attached for cloud IaaS deployments. High-performance NVMe Gen 5 SSDs will provide optimal MongoDB throughput performance and lower latencies.

1.2.3 - Recommended System Resources

Resource requirements depend on the size and resource demands of your MongoDB deployment, but you can use the following general recommendations to get started:

Operating System	Recommended Specifications*
CPU*	8-core processors are the minimum per node for production usage. MongoDB's WiredTiger storage engine architecture can efficiently use multiple CPU cores. Provision an adequate number of CPU cores in proportion to concurrent client connections. In MongoDB Atlas, the number of CPU cores and concurrent client connections is a function of your chosen cluster tier. CPU-heavy operations include (but are not limited to) Page Compression, Data Calculation, Aggregation Framework Operations, and Map Reduce.
RAM*	MongoDB performs best when the application's working set (indexes and most frequently accessed data) fits in memory. RAM size is the most important factor for instance sizing; other optimizations may not significantly improve database performance without sufficient RAM. MongoDB uses system memory for operations such as (but not limited to) Aggregation, Index Traversing, Write Operations, Query Engine, and Connections.
Storage	SATA, PCIe, and NVMe SSDs. Use SSDs for read-heavy applications if the working set no longer fits in memory.
The smaller CPU and RAM configurations apply to virtualized environments or cloud deployments See Performance Best Practices: Hardware and OS Configuration for additional information.	

Table 1-2: Recommended system resource requirements for MongoDB servers

Chapter 2: Hardware Configuration Best Practices

2.1 - Memory Configuration

Proper memory subsystem configuration is crucial for optimum performance. I/O transfers data into or out of memory, so I/O bandwidth can never exceed memory subsystem capabilities. AMD recommends using a symmetric memory population for optimal system performance. Please see the latest version of Memory Population Guidelines for AMD Family 1Ah Models 00h–0Fh and Models 10h–1Fh Socket SP5 Processors - Login required) for additional details.

2.2 - 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 workload.

Table 2-1 describes the BIOS options that most impact MongoDB 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™ 9005 Series Processors* (available from the AMD Documentation Hub) for additional BIOS tuning information.

2.2.1 - BIOS Settings for Maximizing Performance

Name	Value	Description
SMT Control	Enabled	Enabling Simultaneous Multithreading (SMT) allows two hardware threads per core. You must enable AMD x2APIC with support more than 255 threads if you are using a system with dual 64-core AMD EPYC 9005 Series Processors. If you are running dual 64-core processors and your OS does not support AMD x2APIC, then you must disable SMT.
NUMA Node per Socket (NPS)	NPS1	This setting enables a tradeoff between minimizing local memory latency for NUMA-aware or highly parallelizable workloads vs. maximizing per-core memory bandwidth for non-NUMA-friendly workloads by determining the number of NUMA nodes to split the memory channels between. Higher settings reduce memory channels per NUMA node, which lowers both throughput and latency for that NUMA node.
IOMMU	Enabled	Enabling IOMMU allows devices (such as the AMD EPYC processor-integrated SATA controller) to present separate IRQs for each attached device instead of one IRQ for the subsystem. Enabling IOMMU also allows operating systems to provide additional protection for DMA capable I/O devices. If you believe IOMMU is impeding performance, then enable it in BIOS and disable it via OS options, such as `iommu=pt` on the Linux kernel command line. Add `iommu=pt` (passthrough) to the grub file for optimal performance. In passthrough mode, the adapters need not use DMA translation to the memory, which improves performance.
Determinism Control	Power	This setting disables performance determinism and sets the determinism mode to **Power**.
ACPI SRAT L3 Cache as NUMA Domain	Disable	Controls automatic or manual generation of distance information in the ACPI System Locality Information MongoDB (SLIT) and NUMA proximity domains in the System Resource Affinity MongoDB (SRAT). Disabling this option disables reporting each L3 cache as a NUMA domain to the OS.

Table 2-1: Recommended BIOS settings

Chapter 3: Linux Optimizations

Tuning the Linux operating system's Memory and storage subsystems positively impacts MongoDB performance.

3.1 - Memory Subsystem

MongoDB performs best when the application's working set (indexes and most frequently accessed data) fits in memory. RAM size is the most important factor for instance sizing; other optimizations may not significantly improve the performance of the database if there is insufficient RAM.

3.1.1 - Swappiness

Change the kernel swappiness setting to "1" for MongoDB.

The `zone_reclaim_mode` parameter allows you to set aggressive approaches to reclaim memory when a zone runs out of memory. Setting this to zero disables zone reclaim. You must disable `vm.zone_reclaim_mode` when NUMA is enabled.

The `dirty_ratio` is the percentage of total system memory that can hold dirty pages. Most Linux systems default to between 20-30%. Exceeding this limit commits the dirty pages to disk and creates a small pause. You can avoid this hard pause using the `dirty_background_ratio` that defaults 10-15% and tells the kernel to start flushing dirty pages to disk in the background without pausing.

On large-memory (64GB+) database servers, set the ratios in /etc/sysctl.conf as follows:

- `vm.dirty_ratio` = 15
- `vm.dirty_background_ratio` = 5, or possibly less.

```
vm.swappiness=1
vm.zone_reclaim_mode=0
vm.dirty_ratio = 15
vm.dirty_background_ratio = 5
```

3.1.2 - Disabling Transparent Huge Pages (THP)

Linux distributions enable Transparent Huge Pages (THP) by default, which makes the kernel try to allocate memory in large chunks (usually 2MB) instead of 4K. TH) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages. However, database workloads often perform poorly with THP enabled because they tend to have sparse rather than contiguous memory access patterns. Disable THP when running MongoDB on Linux for best performance. Follow the procedures described in the MongoDB Disable Transparent Huge Pages (THP)* documentation to disable THP by creating a service file. On Ubuntu, you can disable THP by executing the command `echo never > /sys/kernel/mm/transparent_hugepage/enabled`.

3.1.3 - NUMA Interleaving

Running MongoDB on a system with Non-Uniform Memory Access (NUMA) can cause a number of operational problems, including slow performance for periods of time, inability to use all available RAM, and high system process usage.

Use the `numactl -interleave` command to configure a memory interleave policy when running MongoDB servers and clients on NUMA hardware. Start the `mongod` process with `numactl -interleave=all`:

```
# numactl -interleave=all /usr/bin/mongod -f /etc/mongod.conf
```

If `system` is in use, then edit `/etc/system/system/multi-user.target.wants/mongod.service` so that the existing `ExecStart` statement reads as follows:

```
# ExecStart=/usr/bin/numactl -interleave=all /usr/bin/mongod -config
/etc/mongod.conf
```

Restart any running `mongod` instances:

```
# sudo systemctl daemon-reload
# sudo systemctl stop mongod
# sudo systemctl start mongod
```

3.2 - Storage Subsystem

MongoDB performs all read and write operations through in-memory data structures. Data is persisted to disk, and queries on data not already in RAM trigger a read from disk. Storage performance is therefore critical.

Most MongoDB disk access patterns do not have sequential properties, and using SSDs may deliver substantial performance gains. Use SSDs for read-heavy applications if the working set no longer fits in memory. AMD EPYC 9005 Series Processors support PCIe® Gen 5 connections that offer twice the I/O bandwidth of PCIe 4.0.

AMD recommends storing MongoDB journal files on a separate disk partition. Most MongoDB deployments should use RAID-10 storage configurations.

MongoDB supports a range of compression options for both documents and indexes. The default snappy compression typically reduces the storage footprint by 50% or more and enables higher IOPs as fewer bits are read from disk. All compression algorithms trade storage efficiency for CPU overhead, and the AMD EPYC SoC is capable of handling the extra load needed to handle compression. Indexes use prefix compression by default, which reduces the in-memory index storage footprint, thereby freeing up more RAM for frequently-accessed documents.

Most Linux systems use the ext4 virtual filesystem by default. AMD recommends using XFS to avoid performance issues observed when using EXT4 with WiredTiger.

3.2.1 - Filesystem Mount Options

Most Linux system use the virtual EXT4 filesystem by default, but XFS is better for MongoDB because it offers slightly better performance for append-only workloads.

Name	Description
noatime	Disables updating the metadata associated with files in the filesystem with an updated access time. This tracking is superfluous because databases maintain their own accesses in their logs.
nobarrier	Disables the filesystem write barrier. Using a write barrier degrades I/O performance by requiring more frequent data flushes.

Table 3-1: XFS file system mount options

Use the following `xfs` filesystem mount options in `/etc/fstab`:

```
/dev/nvme0n1p1     /commitlogdir     xfs     noatime,nobarrier 0 0
```

Set the readahead setting between 8 and 32.

```
# blockdev --setra 32 <storage device path>
```

3.2.2 - I/O Scheduler

Choosing the right disk I/O scheduler algorithm greatly improves performance.

- `deadline scheduler` is essentially a FIFO but does basic reordering and merging within the scheduler queue and guarantees a maximum latency for any given write in its queue.
- `noop` is a less-frequently-used option that works for MongoDB.
- Either `deadline` or `noop` will perform better than CFQ in bare metal installations.

```
# echo deadline > /sys/block/<dev>/queue/scheduler
```

`nr_requests`: The I/O request queue also offers opportunities for boosting performance. This queue determines how many objects can be essentially reordered before being flushed to disk. Longer queues mean better write ordering and fewer head movements a spinning disk will encounter when it starts writing to disk. For NVMe drives, set this to `1024` for `/sys/block/<dev>/queue/nr_requests`.

```
# echo 128 > /sys/block/<dev>/queue/nr_requests
```

Align filesystem I/O down to the physical devices. Unaligned I/O can multiply the number of on-disk writes incurred by any given logical write to your filesystem, which impedes I/O performance. Partition alignment is more critical when using SSD and NVMe drives.

Disk topography determines optimum alignment to a multiple of the physical block size so as to guarantee optimal performance. This example shows creating two partitions using `parted -a optimal` to create partitions that align to a multiple of the physical block size in a way that guarantees optimal performance:

```
# fdisk -l /dev/nvme0n1
# parted /dev/nvme0n1 mklabel gpt
# parted -a optimal /dev/nvme0n1 mkpart primary 0% 50%
# parted -a optimal /dev/nvme0n1 mkpart primary 51% 100%
```

The commit log and data directories (`ssMongoDBs`) should be on different disks. If they both are on the same disk, then place them on separate partitions.

Set the following system limits in `/etc/security/limits.d/99-mongodb-nproc.conf`:

- -f (file size): unlimited
- -t (cpu time): unlimited
- -v (virtual memory): unlimited
- -l (locked-in-memory size): unlimited
- -n (open files): 64000
- -m (memory size): unlimited
- -u (processes/threads): 64000

3.3 - Network Subsystem

Start tuning the adapter TCP/IP stack in Linux by making sure to use the maximum number of both transmit (TX) and receive (RX) ring buffers. Setting initial TX and RX network buffer sizes reduces the amount of post-boot time required for the network to reach an optimal performance state.

See the *Linux® Network Tuning Guide for AMD EPYC™ 9005 Series Processors* (available from the AMD Documentation Hub) for information on configuring the network for systems in a MongoDB cluster and follow the guidelines therein to:

- Tune the TX and RX ring sizes.
- Change the number of interrupts queues to match the cores on the NUMA node which the NIC is collocated and pin those interrupts to the correct processor cores.

You can use the `iperf` utility to stress test the network infrastructure to verify proper setup. Be sure to properly set the OS IOMMU because this has significant performance impact on system performance. This is normally done by setting the IOMMU to pass-through mode by adding the kernel parameter `iommu=pt` on the kernel boot line. If you are using RHEL 8.x, then modify `/etc/default/grub` and run the `grub2-mkconfig` utility.

If you are using Windows, then please see the *Windows® Network Tuning Guide for AMD EPYC™ 9005 Series Processors* (available from the AMD Documentation Hub), for network tuning information.

MongoDB is a distributed database that relies on efficient network transport during query routing and inter- node replication. The snappy compression algorithm compresses MongoDB network traffic across a cluster by up to 80% for significant performance boosts in bandwidth-constrained environments and reducing networking costs.

Enable compression by adding the compressors parameter to the connection string:
`//localhost/?compressors=snappy`

A distributed data store loads the network with read/write requests and replicating data across nodes. The network can become a bottleneck if the cluster includes enough high-performance NVMe drives for storage. The minimum required bandwidth is as low as 1000 Mb/s, but you can ensure sufficient network capacity by using 25GbE or higher Ethernet cards.

Set the following network kernel settings to prevent connections between nodes from timing out:

```
net.core.somaxconn = 4096
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_time = 120
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_keepalive_probes = 6
```

Use the following settings to handle the thousands of concurrent connections used by the MongoDB database:

```
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.core.rmem_default=16777216
net.core.wmem_default=16777216
net.core.optmem_max=40960
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
```

3.4 - tuned-adm Profile

The `tuned-adm` profile `throughput-performance` normally generates the best performance. This profile set up the overall system I/O and memory throughput by configuring the CPU governor, kernel scheduler granularity, disk read ahead, swappiness behavior, and dirty cache write back settings. See `/usr/lib/tuned/throughput-performance/tuned.conf` for these settings.

```
# yum install tuned -y
# systemctl start tuned
# tuned-adm profile throughput-performance
# systemctl enable tuned
# cpupower -c all frequency-set -g performance
```

3.5 - NTP Configuration using Chrony

MongoDB requires an accurate system clock. Chrony is better at keeping the clocks in all MongoDB nodes in the cluster synchronized with the Network Time Protocol than ntpd for most networks than ntpd because:

- It is much faster than NTP at synchronizing to the time server. It can also compensate for fluctuating clock frequencies, such as when a host hibernates or enters sleep mode, or when the clock speed varies due to frequency stepping that slows clock speeds when loads are low.
- It handles intermittent network connections and bandwidth saturation and adjusts for network delays and latency.
- It never stops the clock after the initial update, which ensures stable and consistent time intervals for system services and applications. Please see Using the Chrony suite to configure NTP for detailed information.

```
# timedatectl list-timezones
# timedatectl set-timezone America/Los_Angeles; date # Now setup the Automatic NTP Timing through Chrony
# systemctl status chronyd; systemctl enable chronyd; systemctl start
chronyd; chronyc tracking; hwclock -w; hwclock; date; ps -ef | grep [ch]rony
```

3.6 - Stopping Linux Firewall and SELinux (If Required)

Disable the firewall on servers and SELinux if not required for your testing environment does not need them. IT policies may require firewalls in production environments.

```
# systemctl stop firewalld; systemctl disable firewalld; systemctl list- unit-files | grep -i fire
# cp -p /etc/selinux/config /etc/selinux/config.ORIG; sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /
etc/selinux/config; grep -iw "SELINUX=disabled" /etc/selinux/config
# getenforce; sestatus; setenforce 0; sestatus; getenforce # reboot (This will make the SELinux Disabled
permanently)
```

3.7 - Example RHEL Server Configuration Files

3.7.1 - /etc/default/grub

```
# cat /etc/default/grub
…
…
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb iommu=pt quiet"
…

# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3.7.2 - /etc/rc.local

```
# cat /etc/rc.local
touch /var/lock/subsys/local

cpupower -c all idle-set -d 2
ethtool -G p2p1 rx 4096 tx 4096
/usr/sbin/set_irq_affinity_cpulist.sh 1,5,9,13,17,21,25,29,33,37,41,45,49,53,57,61 p2p1

echo “deadline” > /sys/block/nvme0n1/queue/scheduler
echo 1024 > /sys/block/nvme0n1/queue/nr_requests

echo "never" > /sys/kernel/mm/transparent_hugepage/enabled
echo "never" > /sys/kernel/mm/transparent_hugepage/defrag

echo 0 > /sys/class/block/sda/queue/rotational
echo 8 > /sys/class/block/sda/queue/read_ahead_kb
```

3.7.3 - /etc/sysctl.conf

```
# cat /etc/sysctl.conf
vm.swappiness=1
vm.zone_reclaim_mode=0 vm.dirty_ratio = 15
vm.dirty_background_ratio = 5
vm.max_map_count=128000

net.core.somaxconn = 4096
net.ipv4.tcp_fin_timeout = 30
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_time = 120
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_keepalive_probes = 6

net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.core.rmem_default=16777216
net.core.wmem_default=16777216
net.core.optmem_max=40960
net.ipv4.tcp_rmem=4096 87380 16777216
net.ipv4.tcp_wmem=4096 65536 16777216
```

3.7.4 - /etc/security/limits.conf

Increase the process & file limits only for MongoDB user, hardcoded system-wide:

```
# cat /etc/security/limits.d/99-mongodb-nproc.conf
mongod - fsize unlimited
mongod - cpu unlimited
mongod - as unlimited
mongod - nofile 64000
mongod - rss unlimited
mongod - nproc 64000
mongod - memlock unlimited
```

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 4: Performance Tuning MongoDB Settings

4.1 - MongoDB Architecture

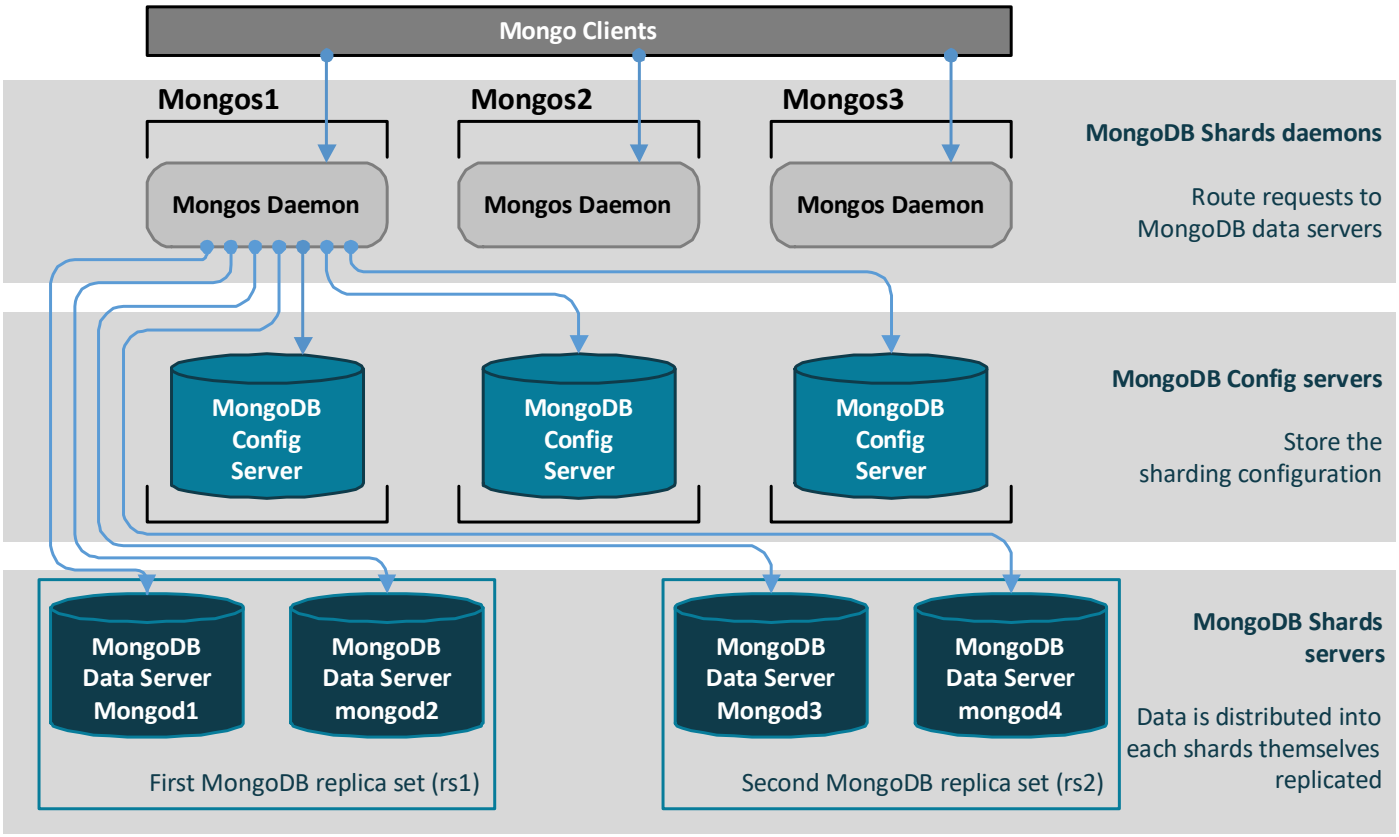

Figure 4-1: MongoDB architecture with Shards and ReplicaSet

4.2 - MongoDB Tuning Best Practices

You must consider MongoDB best practices for the following in order to achieving optimal performance at scale:

- Data modeling and sizing memory (the working set)
- Query patterns and profiling
- Indexing
- Sharding
- Transactions and read/write concerns
- Hardware and OS configuration
- Benchmarking

MongoDB allows flexible schema designs. If you are planning to use sharded clusters for horizontal scaling, then your schema must include a shard key. The shard key affects read and write performance by determining how MongoDB partitions data. Verify that your shard key distributes the load evenly across your shards.

MongoDB prefers indexes. Fields in a document play the role of columns in a SQL database. Like columns, they can be indexed to increase search performance. A missing index forces searching every document within the collection to select the documents requested by the query, which can increase read times.

Working set sizing is another major performance optimization consideration. MongoDB performs best when the application's working set, indexes, and most-frequently-accessed data fits in memory. RAM size is the most important factor for instance sizing; other optimizations may not significantly improve database performance. If price/performance is more of a priority over performance alone, then you can use fast SSDs to complement smaller amounts of RAM. Be sure to test the optimum balance for your workload and SLAs.

If your working set exceeds the RAM of your chosen instance size or server, then consider either moving to a larger instance with more memory or partitioning (sharding) your database across multiple servers. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

There are two ways to optimize performance when the working set size grows:

- **Vertical Scaling:** Increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space.
- **Horizontal Scaling:** Divides the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

To shard a populated collection, the collection must have an index that starts with the shard key. The choice of shard key affects the performance, efficiency, and scalability of a sharded cluster. Selecting the wrong shard key can impede performance on a cluster with the best possible hardware and infrastructure.

MongoDB has built-in replication with auto-elections, which allows setting up a secondary database that can be auto-elected if the primary database becomes unavailable. However, MongoDB requires some setup (and possibly some support) to do replication. MongoDB has replica sets where one member is the primary and all others have a secondary role. The reads and writes are first committed to the primary replica and then replicated to the secondary replicas. MongoDB has a single master. The auto-elect process can take 10 to 40 seconds, during which you cannot write to the replica set.

Creating long-running transactions or attempting to perform an excessive number of operations in a single ACID transaction can pressure the WiredTiger storage engine cache. Consider the following to maintain predictable database performance levels:

- MongoDB allows you to specify the durability guarantee level (write concern) when issuing writes to the database. Write concerns can apply to any operation that executes against the database, regardless of whether it is a regular operation against a single document or wrapped in a multi-document transaction. You can configure write concerns can be configured on a per-connection, per database, per collection, or even per operation basis.

- Read concerns can apply to any query that executes against the database, regardless of whether it is a regular read against a single or set of documents or wrapped in a multi-document read transaction. The read concern configuration can have a significant impact on latency. Supply a `maxTimeMS` value to timeout long-running operations.

4.3 - Locking Performance

MongoDB uses a locking system to ensure data set consistency. If certain operations are long-running or a queue forms, then performance will degrade as requests and operations wait for the lock. Lock-related slowdowns can be intermittent. See the `locks` and `globalLock` sections of the `serverStatus` output to determine whether the lock is affecting performance. Long queries can be caused by:

- Ineffective index use.
- Suboptimal schema design.
- Poor query structure.
- System architecture issues.
- Insufficient RAM causing disk reads.

4.4 - Number of Connections

The number of connections between the applications and the database can sometimes overwhelm the server's ability to handle requests. See the locks and connections sections of the `serverStatus` output. If there are numerous concurrent application requests, then the database may have trouble keeping up with demand. If this is the case, then increase the capacity of your deployment.

- **For read-heavy applications:** Increase the size of your replica set and distribute read operations to secondary members.
- **For write-heavy applications:** Deploy sharding and add one or more shards to a sharded cluster to distribute the load among MongoDB instances.

4.5 - Fine-Grained Telemetry for MongoDB Performance Analysis

Tools like the explain plan, MongoDB Atlas Data Explorer, and MongoDB Query Profiler provide fine- grained telemetry and visibility across all database cluster components.

You can use the MongoDB PY-TPCC* adaption of the TPC-C benchmark for MongoDB (implemented in Python) to evaluate different Atlas tiers or hardware configurations.

MongoDB's `explain ()` method allows you to test queries from your application and shows information about how a query will be, or was, resolved, including:

- The indexes that were used.
- Whether or not the index covered the query.
- Whether an in-memory sort was performed, which indicates an index would be beneficial.
- Number of index entries scanned.

- Number of documents returned, and the number read.
- How long the query took to resolve in milliseconds.
- Which alternative query plans were rejected (when using the `allPlansExecution` mode).

Use MongoDB Query Profiler to expose performance issues by displaying slow-running queries and their key performance statistics directly in the Atlas UI. This utility collects detailed information about operations and commands executed against a running MongoDB instance. All data collected by the profiler is written to the `system.profile` collection. This capped collection resides in the admin database and can be queried for insights. Logging levels can be configured based on the granularity of the data you want to analyze.

MongoDB Atlas features charts, custom dashboards, and automated alerting. It tracks 100+ key database and systems metrics including operations counters, memory, and CPU utilization, replication status, open connections, queues, and any node status.

`mongotop` lets you track how long a MongoDB instance `mongod` takes when reading and writing data. `mongotop` provides statistics on a per-collection level. By default, `mongotop` returns values every second.

The mongostat utility provides a quick status overview of a currently-running `mongod` or `mongos` instance using functionally similar to the UNIX/Linux `vmstat` filesystem utility.

Chapter 5: Resources

- Socket SP5/SP6 Platform NUMA Topology for AMD Family 1Ah Models 00h–0Fh and Models 10h–1Fh - Login required; please review the latest version if multiple versions are present.
- AMD EPYC™ Processor Minimum Operating System (OS) Versions
- From the AMD Documentation Hub:
  - *Windows® Network Tuning Guide for AMD EPYC™ 9005 Series Processors*
  - *Linux® Network Tuning Guide for AMD EPYC™ 9005 Series Processors*
  - *BIOS & Workload Tuning Guide for AMD EPYC™ 9005 Series Processors*
  - *Memory Population Guidelines for AMD EPYC™ 9005 Series Processors*
- MongoDB Supported Platforms*
- MongoDB Production Notes*
- Performance Best Practices: MongoDB Data Modeling and Memory Sizing*
- mongotop*
- MongoDB Operations Checklist*
- Performance Best Practices: Transactions and Read / Write Concerns*
- TPC-C in Python for MongoDB*
- Manage NTP with Chrony*
- Documentation for /proc/sys/vm/*(kernel version 2.6.29)*

THIS PAGE INTENTIONALLY LEFT BLANK.

Chapter 6: Glossary

- **ACPI** - Advanced Configuration and Power Interface
- **BIOS** - Basic Input/Output System
- **BMC** - Baseboard Management Controller
- **CCD** – Core Complex Die
- **CCX** – Core Complexes
- **cTDP** - Configurable Thermal Design Power
- **DIMM** – Dual In-line Memory Module
- **DPC** – DIMMs Per Channel
- **DRAM** – Dynamic Random-Access Memory
- **LLC** - Last Level Cache
- **MDADM** - Multiple Disk and Device Administration
- **NIC** – Network Interface Card
- **NUMA** - Non-Uniform Memory Access
- **PPL** - Package Power Limit
- **OPN** – Orderable Part Number
- **OS** - Operating System
- **SLIT** - System Locality Information Table
- **SMT** - Simultaneous Multithreading
- **SRAT** - System Resource Affinity Table
- **TCO** – Total Cost of Ownership
- **TDP** - Thermal Design Power
- **VM** – Virtual Machine

MongoDB® Tuning Guide for AMD EPYC™ 9005 Processors

PID: 58487

Venkatesan Papavinasam