Storage configuration

Enterprise applications

NetApp
January 12, 2026

Table of Contents

Storage configuration

NFS

PostgreSQL databases can be hosted on NFSv3 or NFSv4 filesystems. The best option depends on factors outside the database.

For example, NFSv4 locking behavior may be preferable in certain clustered environments. (See here for additional details)

Database functionality should otherwise be close to identical, including performance. The only requirement is the use of the `hard` mount option. This is required to ensure soft timeouts do not produce unrecoverable IO errors.

If NFSv4 is chosen as a protocol, NetApp recommends using NFSv4.1. There are some functional enhancements to the NFSv4 protocol in NFSv4.1 that improve resiliency over NFSv4.0.

Use the following mount options for general database workloads:

```
rw,hard,nointr,bg,vers=[3|4],proto=tcp,rsize=65536,wsize=65536
```

If heavy sequential IO is expected, the NFS transfer sizes can be increased as described in the following section.

NFS Transfer Sizes

By default, ONTAP limits NFS I/O sizes to 64K.

Random I/O with an most applications and databases uses a much smaller block size which is well below the 64K maximum. Large-block I/O is usually parallelized, so the 64K maximum is also not a limitation to obtaining maximum bandwidth.

There are some workloads where the 64K maximum does create a limitation. In particular, single-threaded operations such as backup or recovery operation or a database full table scan run faster and more efficiently if the database can perform fewer but larger I/Os. The optimum I/O handling size for ONTAP is 256K.

The maximum transfer size for a given ONTAP SVM can be changed as follows:

```
Cluster01::> set advanced
Warning: These advanced commands are potentially dangerous; use them only
when directed to do so by NetApp personnel.
Do you want to continue? {y|n}: y
Cluster01::*> nfs server modify -vserver vserver1 -tcp-max-xfer-size
262144
Cluster01::*>
```

> Never decrease the maximum allowable transfer size on ONTAP below the value of rsize/wsize of currently mounted NFS file systems. This can create hangs or even data corruption with some operating systems. For example, if NFS clients are currently set at an rsize/wsize of 65536, then the ONTAP maximum transfer size could be adjusted between 65536 and 1048576 with no effect because the clients themselves are limited. Reducing the maximum transfer size below 65536 can damage availability or data.

Once the transfer size is increased at the ONTAP level, the following mount options would be used:

```
rw,hard,nointr,bg,vers=[3|4],proto=tcp,rsize=262144,wsize=262144
```

NFSv3 TCP Slot Tables

If NFSv3 is used with Linux, it is critical to properly set the TCP slot tables.

TCP slot tables are the NFSv3 equivalent of host bus adapter (HBA) queue depth. These tables control the number of NFS operations that can be outstanding at any one time. The default value is usually 16, which is far too low for optimum performance. The opposite problem occurs on newer Linux kernels, which can automatically increase the TCP slot table limit to a level that saturates the NFS server with requests.

For optimum performance and to prevent performance problems, adjust the kernel parameters that control the TCP slot tables.

Run the `sysctl -a | grep tcp.*.slot_table` command, and observe the following parameters:

```
# sysctl -a | grep tcp.*.slot_table
sunrpc.tcp_max_slot_table_entries = 128
sunrpc.tcp_slot_table_entries = 128
```

All Linux systems should include `sunrpc.tcp_slot_table_entries`, but only some include `sunrpc.tcp_max_slot_table_entries`. They should both be set to 128.

> Failure to set these parameters may have significant effects on performance. In some cases, performance is limited because the linux OS is not issuing sufficient I/O. In other cases, I/O latencies increases as the linux OS attempts to issue more I/O than can be serviced.

SAN

PostgreSQL databases with SAN are generally hosted on xfs filesystems, but others can be used if supported by the OS vendor

While a single LUN can generally support up to 100K IOPS, IO-intensive databases generally require the use of LVM with striping.

LVM Striping

Before the era of flash drives, striping was used to help overcome the performance limitations of spinning drives. For example, if an OS needs to perform a 1MB read operation, reading that 1MB of data from a single drive would require a lot of drive head seeking and reading as the 1MB is slowly transferred. If that 1MB of data was striped across 8 LUNs, the OS could issue eight 128K read operations in parallel and reduce the time required to complete the 1MB transfer.

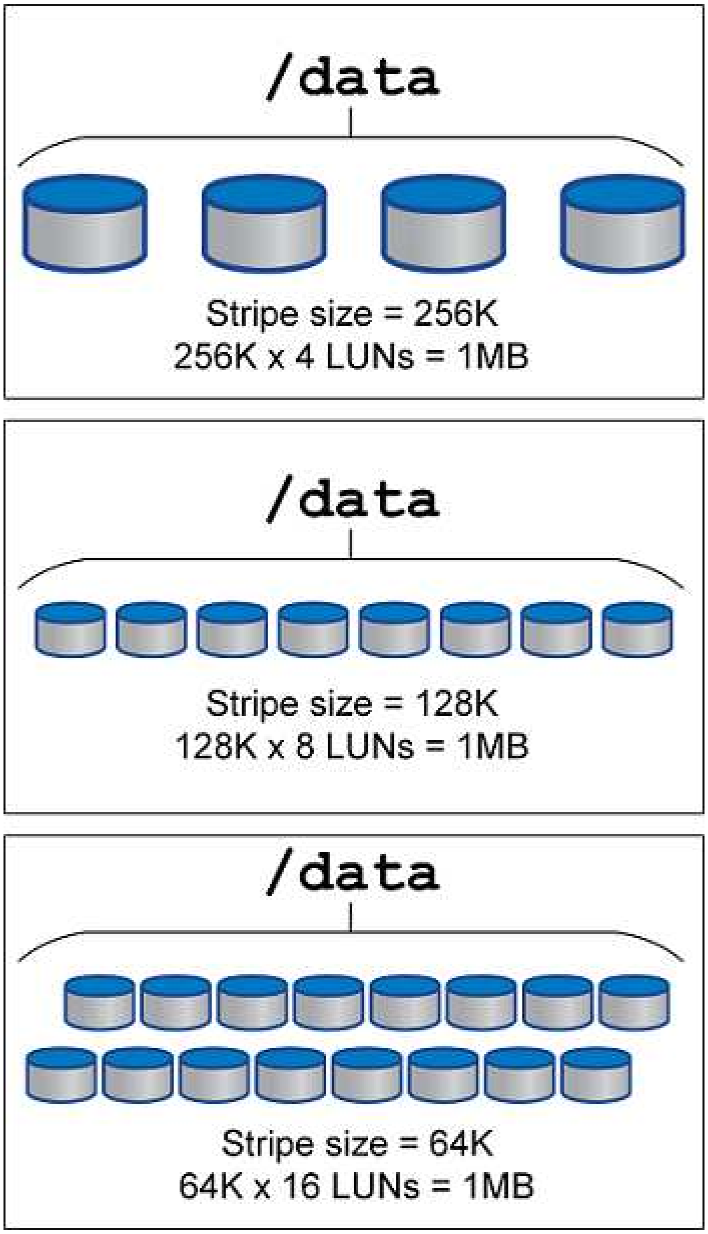
Striping with spinning drives was more difficult because the I/O pattern had to be known in advance. If the striping wasn't correctly tuned for the true I/O patterns, striped configurations could damage performance. With Oracle databases, and especially with all-flash configurations, striping is much easier to configure and has been proven to dramatically improve performance.

Logical volume managers such as Oracle ASM stripe by default, but native OS LVM do not. Some of them bond multiple LUNs together as a concatenated device, which results in datafiles that exist on one and only one LUN device. This causes hot spots. Other LVM implementations default to distributed extents. This is similar to striping, but it's coarser. The LUNs in the volume group are sliced into large pieces, called extents and typically measured in many megabytes, and the logical volumes are then distributed across those extents. The result is random I/O against a file should be well distributed across LUNs, but sequential I/O operations are not as efficient as they could be.

Performance-intensive application I/O is nearly always either (a) in units of the basic block size or (b) one megabyte.

The primary goal of a striped configuration is to ensure that single-file I/O can be performed as a single unit, and multiblock I/Os, which should be 1MB in size, can be parallelized evenly across all LUNs in the striped volume. This means that the stripe size must not be smaller than the database block size, and the stripe size multiplied by the number of LUNs should be 1MB.

The following figure shows three possible options for stripe size and width tuning. The number of LUNs is selected to meet performance requirements as described above, but in all cases the total data within a single stripe is 1MB.


/data
Stripe size = 256K
256K x 4 LUNs = 1MB
/data
Stripe size = 128K
128K x 8 LUNs = 1MB
/data
Stripe size = 64K
64K x 16 LUNs = 1MB