MySQL

Enterprise applications

NetApp
April 25, 2024

Table of Contents

MySQL

MySQL databases on ONTAP

MySQL and its variants, including MariaDb and Percona MySQL, is the world’s most popular database.

> This documentation on ONTAP and the MySQL database replaces the previously published *TR-4722: MySQL database on ONTAP best practices.*

ONTAP is an ideal platform for MySQL database because ONTAP is literally designed for databases. Numerous features such as random IO latency optimizations to advanced quality of service (QoS) to basic FlexClone functionality were created specifically to address the needs of database workloads.

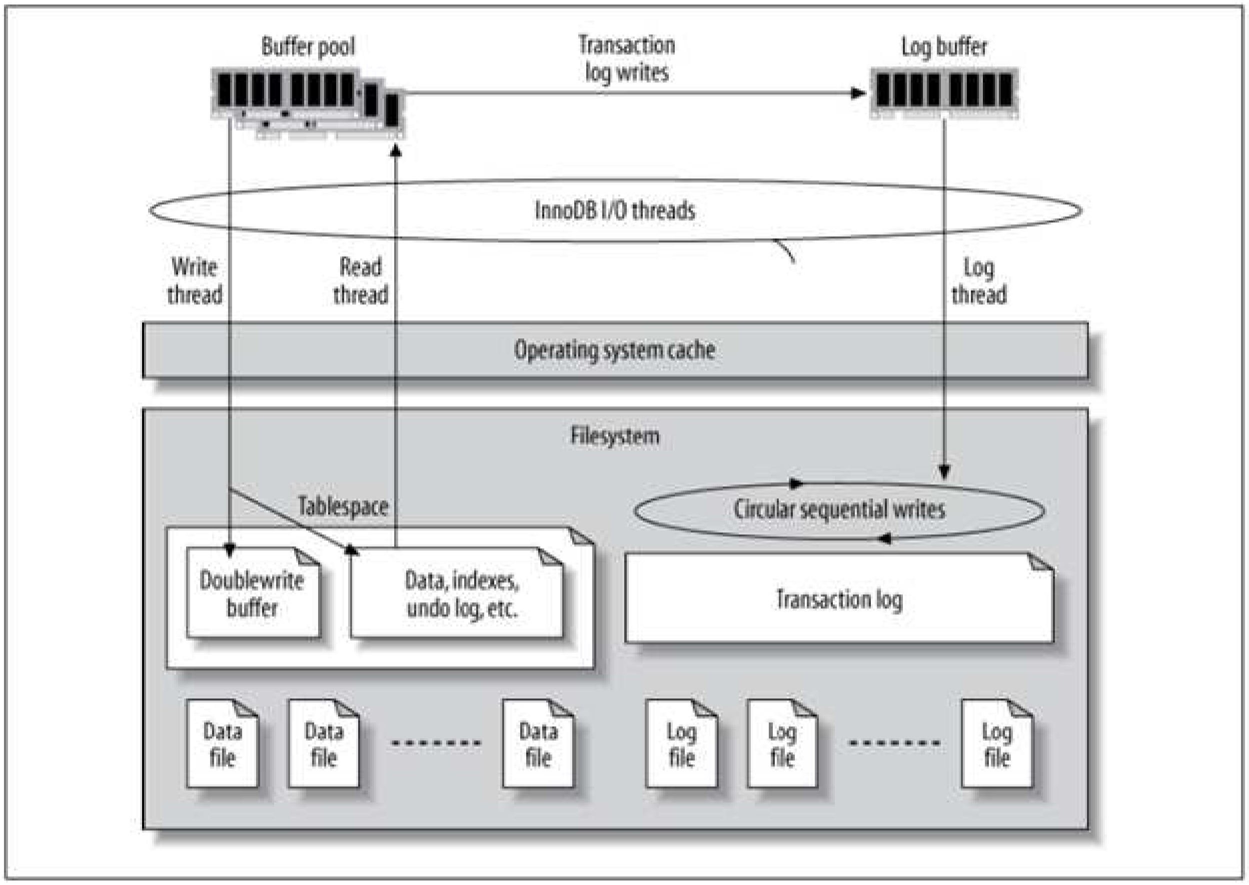
Additional features such as nondisruptive upgrades, (including storage replacement) ensure that your critical databases remain available. You can also have instant disaster recovery for large environments through MetroCluster, or select databases using SnapMirror active sync.

Most importantly, ONTAP delivers unmatched performance with the ability to size the solution for your unique needs. Our high-end systems can deliver over 1M IOPS with latencies measured in microseconds, but if you only need 100K IOPS you can right-size your storage solution with a smaller controller that still runs the exact same storage operating system.

Database configuration

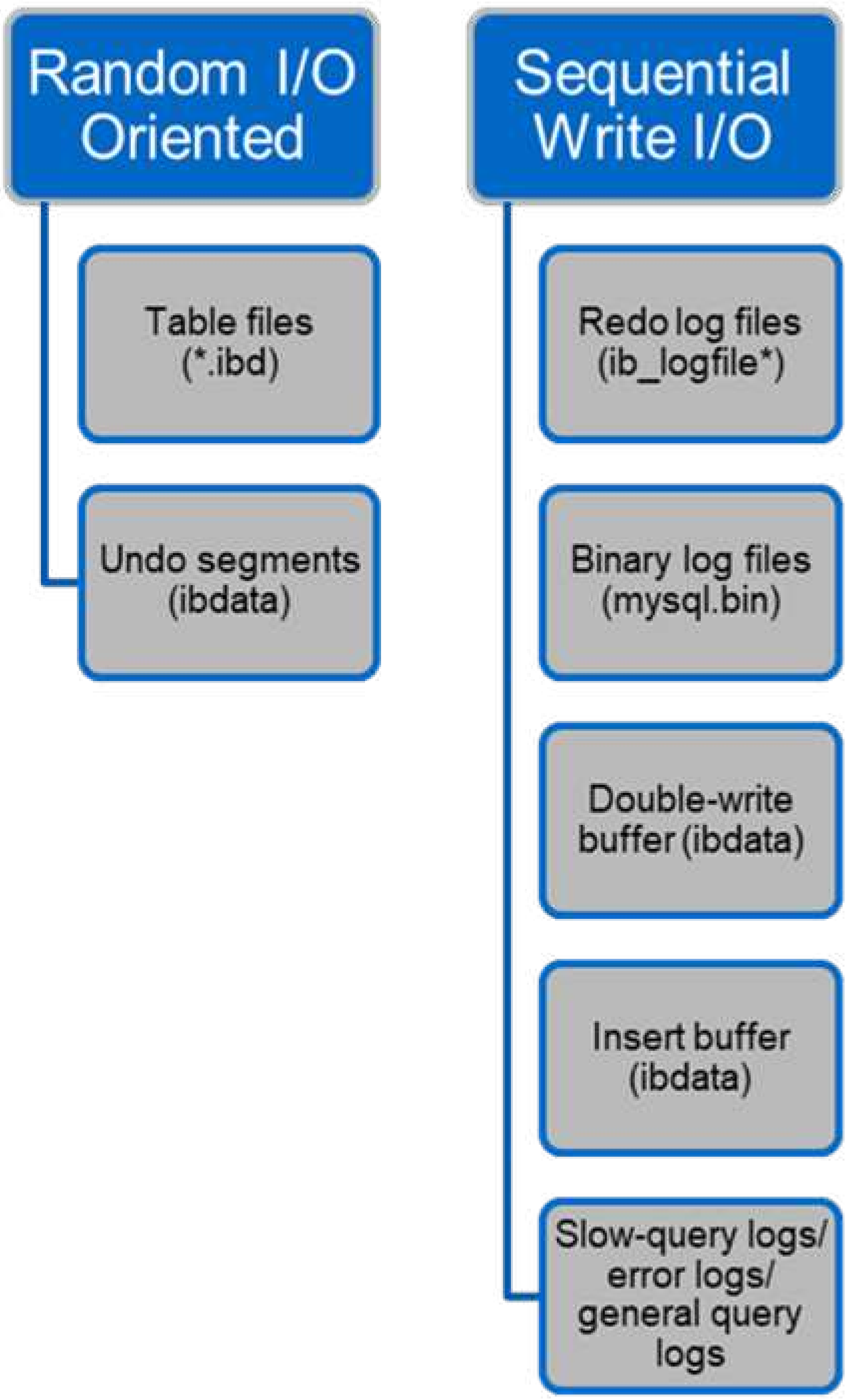
MySQL and InnoDB

InnoDB acts as the middle layer between storage and the MySQL server, it stores the data to the drives.



MySQL I/O is categorized into two types:

- Random file I/O
- Sequential file I/O


Random I/O Oriented
Table files (*.ibd)
Undo segments (ibdata)
Sequential Write I/O
Redo log files (ib_logfile*)
Binary log files (mysql.bin)
Double-write buffer (ibdata)
Insert buffer (ibdata)
Slow-query logs/ error logs/ general query logs

Data files are randomly read and overwritten, which results in high IOPS. Therefore, SSD storage is recommended.

Redo log files and binary log files are transactional logs. They are sequentially written, so you can get good performance on HDD with the write cache. A sequential read happens on recovery, but it rarely causes a performance problem, because log file size is usually smaller than data files, and sequential reads are faster than random reads (occurring on data files).

The double-write buffer is a special feature of InnoDB. InnoDB first writes flushed pages to the double-write buffer and then writes the pages to their correct positions on the data files. This process prevents page corruption. Without the double-write buffer, the page might become corrupted if a power failure occurs during the write-to-drives process. Because writing to the double-write buffer is sequential, it is highly optimized for HDDs. Sequential reads occur on recovery.

Because ONTAP NVRAM already provides write protection, double-write buffering is not required. MySQL has a parameter, `skip_innodb_doublewrite`, to disable the double-write buffer. This feature can substantially improve performance.

The insert buffer is also a special feature of InnoDB. If non-unique secondary index blocks are not in memory, InnoDB inserts entries into the insert buffer to avoid random I/O operations. Periodically, the insert buffer is merged into the secondary index trees in the database. The insert buffer reduces the number of I/O operations by merging I/O requests to the same block; random I/O operations can be sequential. The insert buffer is also highly optimized for HDDs. Both sequential writes and reads occur during normal operations.

Undo segments are random I/O oriented. To guarantee multi-version concurrency (MVCC), InnoDB must register old images in the undo segments. Reading previous images from the undo segments requires random reads. If you run a long transaction with repeatable reads (such as mysqldump—single transaction) or run a long query, random reads can occur. Therefore, storing undo segments on SSDs is better in this instance. If you run only short transactions or queries, the random reads are not an issue.

NetApp recommends the following storage design layout because of the InnoDB I/O characteristics.

- One volume to store random and sequential I/O-oriented files of MySQL
- Another volume to store purely sequential I/O-oriented files of MySQL

This layout also helps you design data protection policies and strategies.

MySQL configuration parameters

NetApp recommends a few important MySQL configuration parameters to obtain optimal performance.

Parameters	Values
innodb_log_file_size	256M
innodb_flush_log_at_trx_commit	2
innodb_doublewrite	0
innodb_flush_method	fsync
innodb_buffer_pool_size	11G
innodb_io_capacity	8192
innodb_buffer_pool_instances	8
innodb_lru_scan_depth	8192
open_file_limit	65535

To set the parameters described in this section, you must change them in the MySQL configuration file (my.cnf). The NetApp best practices are a result of tests performed in-house.

innodb_log_file_size

Selecting the right size for the InnoDB log file size is important for the write operations and for having a decent recovery time after a server crash.

Because so many transactions are logged in to the file, the log file size is important for write operations. When records are modified, the change is not immediately written back to the tablespace. Instead, the change is recorded at the end of the log file and the page is marked as dirty. InnoDB uses its log to convert the random I/O into sequential I/O.

When the log is full, the dirty page is written out to the tablespace in sequence to free up space in the log file. For example, suppose a server crashes in the middle of a transaction, and the write operations are only recorded in the log file. Before the server can go live again, it must go through a recovery phase in which the changes recorded in the log file are replayed. The more entries that are in the log file, the longer it takes for the server to recover.

In this example, the log file size affects both the recovery time and the write performance. When choosing the right number for the log file size, balance the recovery time against write performance. Typically, anything between 128M and 512M is a good value.

innodb_flush_log_at_trx_commit

When there is a change to the data, the change is not immediately written to storage.

Instead, the data is recorded in a log buffer, which is a portion of memory that InnoDB allocates to buffer changes that are recorded in the log file. InnoDB flushes the buffer to the log file when a transaction is committed, when the buffer gets full, or once per second, whichever event happens first. The configuration variable that controls this process is innodb_flush_log_at_trx_commit. The value options include:

- When you set `innodb_flush_log_trx_at_commit=0`, InnoDB writes the modified data (in the InnoDB buffer pool) to the log file (ib_logfile) and flushes the log file (write to storage) every second. However, it does not do anything when the transaction is committed. If there is a power failure or system crash, none of the unflushed data is recoverable because it is not written to either the log file or drives.
- When you set `innodb_flush_log_trx_commit=1`, InnoDB writes the log buffer to the transaction log and flushes to durable storage for every transaction. For example, for all transaction commits, InnoDB writes to the log and then writes to storage. Slower storage negatively affects performance; for example, the number of InnoDB transactions per second is reduced.
- When you set `innodb_flush_log_trx_commit=2`, InnoDB writes the log buffer to the log file at every commit; however, it doesn't write data to storage. InnoDB flushes data once every second. Even if there is a power failure or system crash, option 2 data is available in the log file and is recoverable.

If performance is the main goal, set the value to 2. Since InnoDB writes to the drives once per second, not for every transaction commit, performance improves dramatically. If a power failure or crash occurs, data can be recovered from the transaction log.

If data safety is the main goal, set the value to 1 so that for every transaction commit, InnoDB flushes to the drives. However, performance might be affected.

NetApp recommends set the innodb_flush_log_trx_commit value to 2 for better performance.

innodb_doublewrite

When `innodb_doublewrite` is enabled (the default), InnoDB stores all data twice: first to the double-write buffer and then to the actual data files.

You can turn off this parameter with `--skip-innodb_doublewrite` for benchmarks or when you're more concerned with top performance than data integrity or possible failures. InnoDB uses a file flush technique called double-write. Before it writes pages to the data files, InnoDB writes them to a contiguous area called the double-write buffer. After the write and the flush to the double-write buffer are complete, InnoDB writes the pages to their proper positions in the data file. If the operating system or a mysqld process crashes during a page write, InnoDB can later find a good copy of the page from the double-write buffer during crash recovery.

NetApp recommends disabling the double-write buffer. ONTAP NVRAM serves the same function. Double-buffering will unnecessarily damage performance.

innodb_buffer_pool_size

The InnoDB buffer pool is the most important part of any tuning activity.

InnoDB relies heavily on the buffer pool for caching indexes and rowing the data, the adaptive hash index, the insert buffer, and many other data structures used internally. The buffer pool also buffers changes to data so that write operations don't have to be performed immediately to storage, thus improving performance. The buffer pool is an integral part of InnoDB and its size must be adjusted accordingly. Consider the following factors when setting the buffer pool size:

- For a dedicated InnoDB-only machine, set the buffer pool size to 80% or more of available RAM.
- If it's not a MySQL dedicated server, set the size to 50% of RAM.

innodb_flush_method

The innodb_flush_method parameter specifies how InnoDB opens and flushes the log and data files.

Optimizations

In InnoDB optimization, setting this parameter tweaks the database performance when applicable.

The following options are for flushing the files through InnoDB:

- `fsync`. InnoDB uses the `fsync()` system call to flush both the data and log files. This option is the default setting.
- `O_DSYNC`. InnoDB uses the `O_DSYNC` option to open and flush the log files and fsync() to flush the data files. InnoDB does not use `O_DSYNC` directly, because there have been problems with it on many varieties of UNIX.
- `O_DIRECT`. InnoDB uses the `O_DIRECT` option (or `directio()` on Solaris) to open the data files and uses `fsync()` to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.
- `O_DIRECT_NO_FSYNC`. InnoDB uses the `O_DIRECT` option during flushing I/O; however, it skips the `fsync()` system call afterward. This option is unsuitable for some types of file systems (for example, XFS). If you are not sure if your file system requires an `fsync()` system call—for example, to preserve all

file metadata—use the `O_DIRECT` option instead.

Observation

In the NetApp lab tests, the `fsync` default option was used on NFS and SAN, and it was a great performance improviser compared to `O_DIRECT`. While using the flush method as `O_DIRECT` with ONTAP, we observed that the client writes a lot of single-byte writes at the border of the 4096 block in serial fashion. These writes increased latency over the network and degraded performance.

innodb_io_capacity

In the InnoDB plug-in, a new parameter called innodb_io_capacity was added from MySQL 5.7.

It controls the maximum number of IOPS that InnoDB performs (which includes the flushing rate of dirty pages as well as the insert buffer [ibuf] batch size). The innodb_io_capacity parameter sets an upper limit on IOPS by InnoDB background tasks, such as flushing pages from the buffer pool and merging data from the change buffer.

Set the innodb_io_capacity parameter to the approximate number of I/O operations that the system can perform per second. Ideally, keep the setting as low as possible, but not so low that background activities slow down. If the setting is too high, data is removed from the buffer pool and insert buffer too quickly for caching to provide a significant benefit.

NetApp recommends that if using this setting over NFS, analyzing the test result of IOPS (SysBench/Fio) and set the parameter accordingly. Use the smallest value possible for flushing and purging to keep up unless you see more modified or dirty pages than you want in the InnoDB buffer pool.

Do not use extreme values such as 20,000 or more unless you’ve proved that lower values are not sufficient for your workload.

The Innodb_IO_capacity parameter regulates flushing rates and related I/O.

You can seriously harm performance by setting this parameter or the innodb_io_capacity_max parameter too high and wasting I/O operations with premature flushing.

innodb_lru_scan_depth

The `innodb_lru_scan_depth` parameter influences the algorithms and heuristics of the flush operation for the InnoDB buffer pool.

This parameter is primarily of interest to performance experts tuning I/O-intensive workloads. For each buffer pool instance, this parameter specifies how far down in the least recently used (LRU) page list the page cleaner thread should continue scanning, looking for dirty pages to flush. This background operation is performed once per second.

You can adjust the value up or down to minimize the number of free pages. Don’t set the value much higher than needed, because the scans can have a significant performance cost. Also, consider adjusting this parameter when changing the number of buffer pool instances, because `innodb_lru_scan_depth * innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

A setting smaller than the default is suitable for most workloads. Consider increasing the value only if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially if you have a large buffer pool.

open_file_limits

The `open_file_limits` parameter determines the number of files that the operating system permits mysqld to open.

The value of this parameter at run time is the real value permitted by the system and might be different from the value you specify at server startup. The value is 0 on systems where MySQL cannot change the number of open files. The effective `open_files_limit` value is based on the value that is specified at the system startup (if any) and the values of `max_connections` and `table_open_cache` by using these formulas:

- 10 + `max_connections` + (`table_open_cache` x 2)
- `max_connections` x 5
- Operating system limit if positive
- If the operating system limit is infinity: `open_files_limit` value is specified at startup; 5,000 if none

The server attempts to obtain the number of file descriptors using the maximum of these four values. If that many descriptors cannot be obtained, the server attempts to obtain as many as the system will permit.

Host configuration

MySQL containerzation

Containerization of MySQL databases is becoming more prevalent.

Low-level container management is almost always performed through Docker. Container management platforms such as OpenShift and Kubernetes make management of large container environments even simpler. The benefits of containerization include lower costs, because there is no need to license a hypervisor. Also, containers allow multiple databases to run isolated from one another while sharing the same underlying kernel and operating system. Containers can be provisioned in microseconds.

NetApp offers Astra Trident to provide advanced management capabilities of storage. For example, Astra Trident allows a container created in Kubernetes to automatically provision its storage on the appropriate tier, apply export policies, set snapshot policies, and even clone one container to another. For additional information, see the Astra Trident documentation.

MySQL and NFSv3 slot tables

NFSv3 performance on Linux depends on a parameter called `tcp_max_slot_table_entries`.

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.

Caution
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.

I/O schedulers and MySQL

The Linux kernel allows low-level control over the way that I/O to block devices is scheduled.

The defaults on various distributions of Linux vary considerably. MySQL recommends that you use `NOOP` or a `deadline` I/O scheduler with native asynchronous I/O (AIO) on Linux. In general, NetApp customers and internal testing show better results with NoOps.

MySQL's InnoDB storage engine uses the asynchronous I/O subsystem (native AIO) on Linux to perform read-ahead and write requests for data file pages. This behavior is controlled by the `innodb_use_native_aio` configuration option, which is enabled by default. With native AIO, the type of I/O scheduler has greater influence on I/O performance. Conduct benchmarks to determine which I/O scheduler provides the best results for your workload and environment.

See the relevant Linux and MySQL documentation for instructions on configuring the I/O scheduler.

MySQL file descriptors

To run, the MySQL server needs file descriptors, and the default values are not sufficient.

It uses them to open new connections, store tables in the cache, create temporary tables to resolve complicated queries, and access persistent ones. If mysqld is not able to open new files when needed, it can stop functioning correctly. A common symptom of this issue is error 24, “Too many open files.” The number of file descriptors mysqld can open simultaneously is defined by the `open_files_limit` option set in the configuration file (`/etc/my.cnf`). But `open_files_limit` also depends on the limits of the operating system. This dependency makes setting the variable more complicated.

MySQL cannot set its `open_files_limit` option higher than what is specified under `ulimit 'open files'`. Therefore, you need to explicitly set these limits at the operating system level to allow MySQL to open files as needed. There are two ways to check the file limit in Linux:

- The `ulimit` command quickly gives you a detailed description of the parameters being allowed or locked. The changes made by running this command are not permanent and will erase after a system reboot.
- Changes to the `/etc/security/limit.conf` file are permanent and are not affected by a system reboot.

Make sure to change both the hard and soft limits for user mysql. The following excerpts are from the configuration:

```
mysql hard nofile 65535
mysql soft nofile 65353
```

In parallel, update the same configuration in `my.cnf` to fully use the open file limits.

Storage configuration

MySQL with NFS

The MySQL documentation recommends that you use NFSv4 for NAS deployments.

ONTAP NFS transfer sizes

By default, ONTAP will limit NFS IO sizes to 64K. Random IO with an MySQL database uses a much smaller block size which is well below the 64K maximum. Large-block IO is usually parallelized, so the 64K maximum is also not a limitation.

There are some workloads where the 64K maximum does create a limitation. In particular, single-threaded operations such as full table scan backup operations will run faster and more efficiently if the database can perform fewer but larger IO’s. The optimum IO handling size for ONTAP with database workloads is 256K. The NFS mount options listed for specific operating systems below have been updated from 64K to 256K accordingly.

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
```

Never decrease the maximum allowable transfer size on ONTAP below the value of rsize/wsize of currently mounted NFS filesystems. This can create hangs or even data corruption with some operating systems. For example, if NFS clients are currently set at a 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.

NetApp recommends

Setting the following NFSv4 fstab (/etc/fstab) setting:

```
nfs4 rw,
hard,nointr,bg,vers=4,proto=tcp,noatime,rsize=262144,wsize=262144
```

A common issue with NFSv3 was the locked InnoDB log files after a power outage. Using time or switching log files solved this issue. However, NFSv4 has locking operations and keeps track of open files and delegations.

MySQL with SAN

There are two options to configure MySQL with SAN using the usual two-volume model.

Smaller databases can be placed on a pair of standard LUNs as long as the I/O and capacity demands are within the limits of a single LUN file system. For example, a database that requires approximately 2K random IOPS can be hosted on a single file system on a single LUN. Likewise, a database that is only 100GB in size would fit on a single LUN without creating a management problem.

Larger databases require multiple LUNs. For example, a database that requires 100K IOPS would most likely need at least eight LUNs. A single LUN would become a bottleneck because of the inadequate number of SCSI channels to drives. A 10TB database would similarly be difficult to manage on a single 10TB LUN. Logical volume managers are designed to bond the performance and capacity capabilities of multiple LUNs together to improve performance and manageability.

In both cases, a pair of ONTAP volumes should be sufficient. With a simple configuration, the data file LUN would be placed in a dedicated volume, as would the log LUN. With a logical volume manager configuration, all the LUNs in the data file volume group would be in a dedicated volume, and the LUNs of the log volume group would be in a second dedicated volume.

NetApp recommends using two file systems for MySQL deployments on SAN:

- The first file system stores all MySQL data including tablespace, data, and index.
- The second file system stores all logs (binary logs, slow logs, and transaction logs).

There are multiple reasons for separating data in this manner, including:

- The I/O patterns of data files and log files differ. Separating them would allow more options with QoS controls.
- Optimal use of Snapshot technology requires the ability to independently restore the data files. Commingling data files with log files interferes with data file restoration.
- NetApp SnapMirror technology can be used to provide a simple, low-RPO disaster recovery capability for a database; however, it requires different replication schedules for the data files and logs.

Use this basic two-volume layout to future-proof the solution so that all ONTAP features can be used if needed.

NetApp recommends formatting your drive with the ext4 file system because of the following features:

- Extended approach to block management features used in the journaling file system (JFS) and delayed allocation features of the extended file system (XFS).
- Ext4 permits file systems of up to 1 exbibyte (2^60 bytes) and files of up to 16 tebibytes (16 * 2^40 bytes). In contrast, the ext3 file system supports only a maximum file system size of 16TB and a maximum file size of 2TB.
- In ext4 file systems, multiblock allocation (mballoc) allocates multiple blocks for a file in a single operation, instead of allocating them one by one, as in ext3. This configuration reduces the overhead of calling the block allocator several times, and it optimizes the allocation of memory.
- Although XFS is the default for many Linux distributions, it manages metadata differently and is not suitable for some MySQL configurations.

NetApp recommends using 4k block size options with the mkfs utility to align with existing block LUN size.

```
mkfs.ext4 -b 4096
```

NetApp LUNs store data in 4KB physical blocks, which yields eight 512-byte logical blocks.

If you do not set up the same block size, I/O will not be aligned with physical blocks correctly and could write in two different drives in a RAID group, resulting in latency.

It is important that you align I/O for smooth read/write operations. However, when the I/O begins at a logical block that is not at the start of a physical block, the I/O is misaligned. I/O operations are aligned only when they begin at a logical block—the first logical block in a physical block.