Database configuration

Enterprise applications

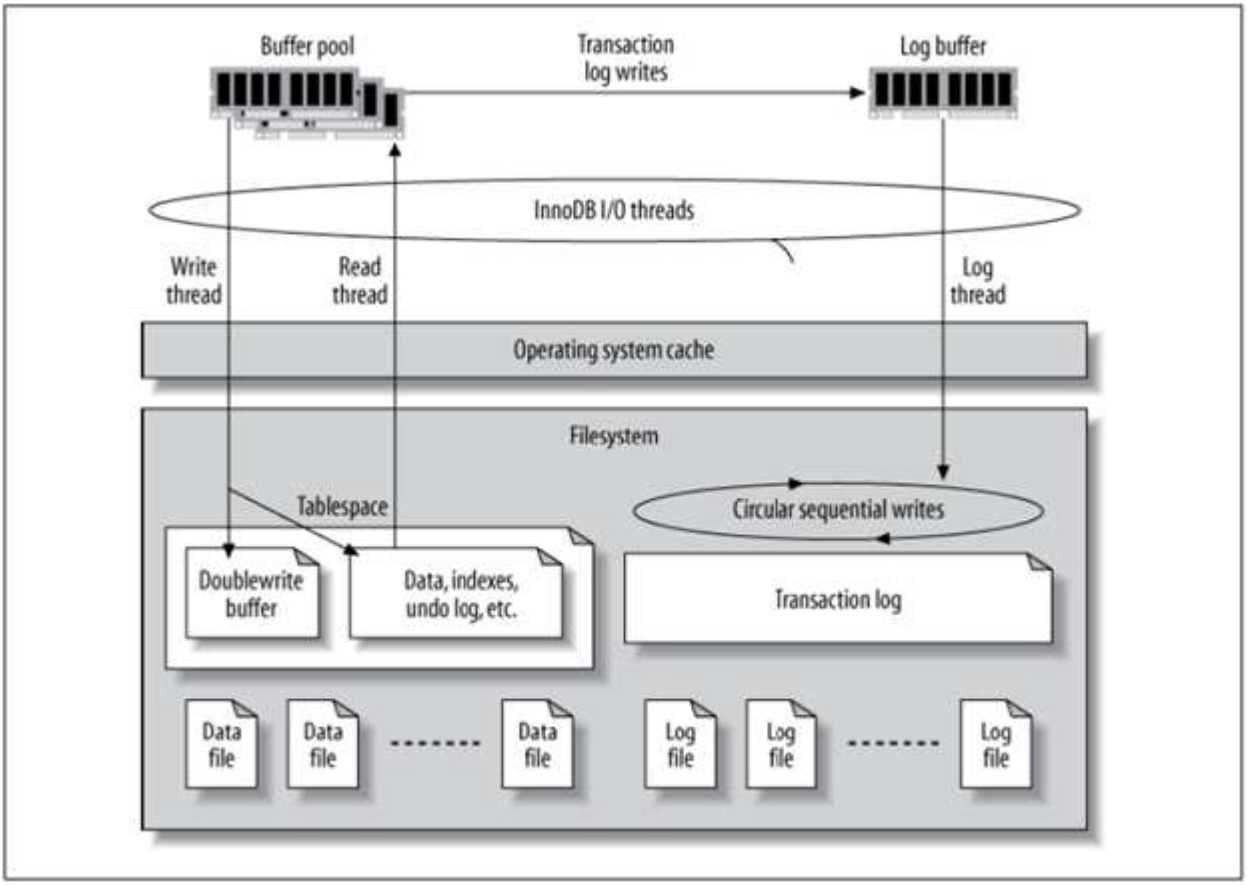
NetApp
March 13, 2024

Table of Contents

Database configuration

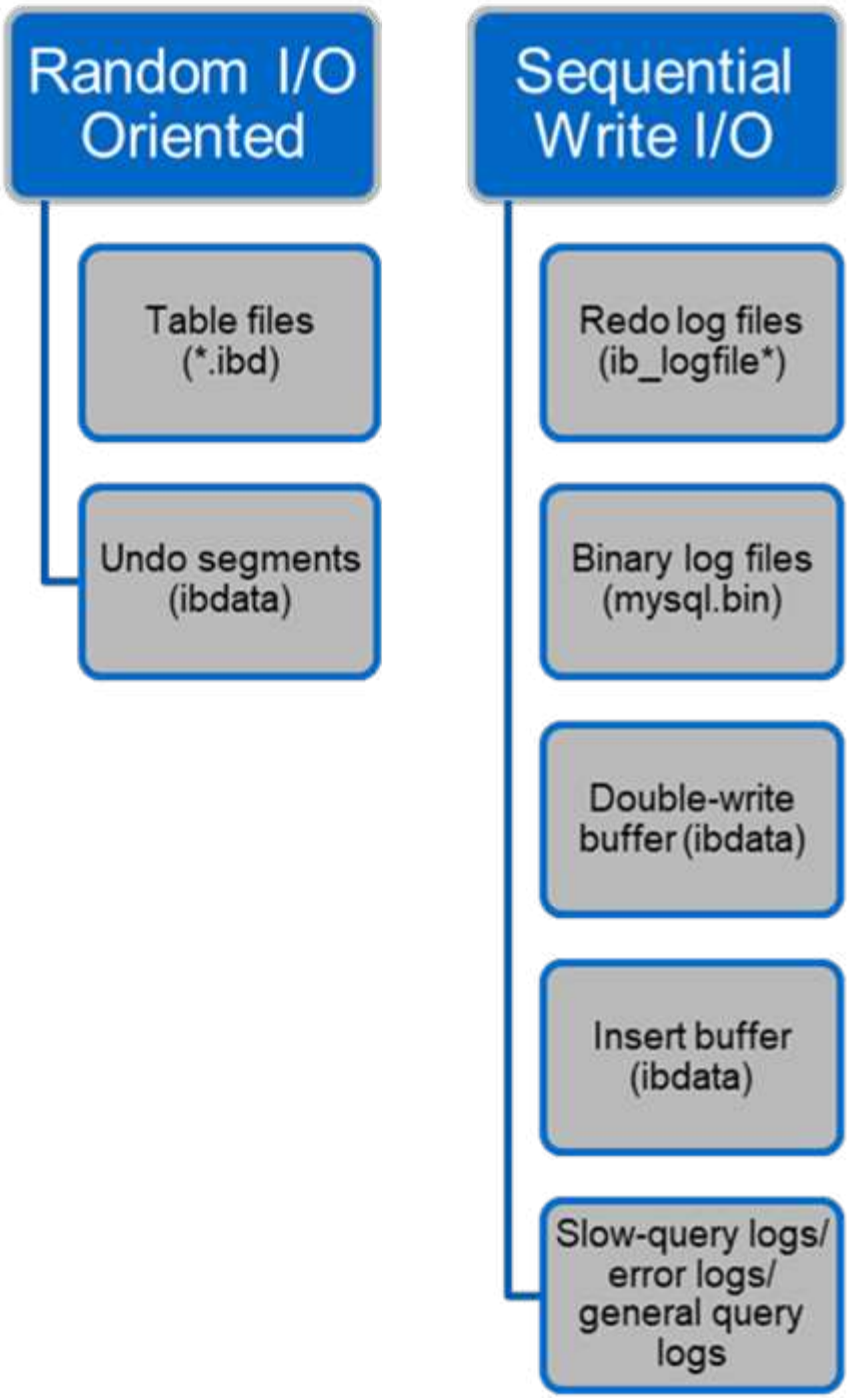
MySQL and InnoDB

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



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.

Copyright information

Trademark information