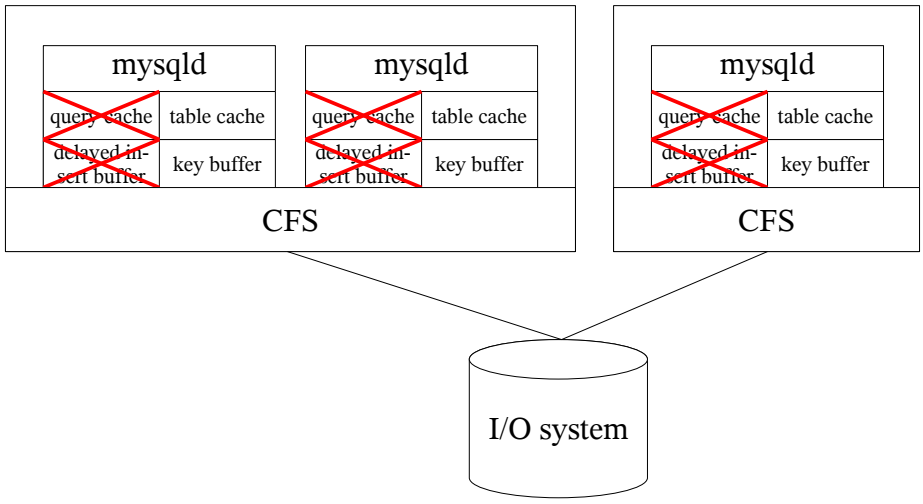
MySQL Active - Active Clustering [1]

It is possible to use an active - active shared-disk cluster in MySQL in some cases. For doing this you have to fulfill the following requirements:

- Works with MyISAM tables only.
- POSIX-locking compliant cluster file system on the device (such as OCFS2 or GFS).
- External locking must be enabled.
- The MySQL query cache must be turned off.
- The MySQL delay key write must be turned off.
- OS where file locking is supported in MySQL.



First of all, MyISAM is the only storage engine in MySQL that has any support for a shared-disk active-active setup. The way MyISAM handles this is by effectively 'passing' down the locks to the file system to enforce. So when there is a shared lock on a table , the shared lock will also set a shared lock in the file system on the files that represent the table. The same thing applies with exclusive lock as well.

In order for this to work, a cluster file system setup that supports these locking characteristics (POSIX standard locking) is required. Two commonly used file systems under Linux that support this locking are OCFS2 (Oracle cluster file system version 2 [2]) and GFS (global file system [4, 5]) from Red Hat. They are not the only possible file systems that could be used, but they are both freely available for use under the GPL. OCFS2 is actually included in the default Linux 2.6 kernel as of 2.6.16. Normal file systems do not support the distributed locking required for this to work.

Now to get MyISAM to make use of the locking on the file system, it is necessary to set the skip_external_locking variable to OFF (`skip_external_locking = 0`, which means enable external locking). This will cause MyISAM to propagate the locks down to the file system for it to manage them.

The other big issue that has to be dealt with are caches. Caution is required when using caches because MySQL does not do any cache synchronization.

First cache to examine is the MySQL query cache. If using this cache, then it is possible to get incorrect results. This will occur when a query is cached on one server, the data then changes through

a different server, the query cache is now out of data and serving incorrect results. There are two ways that this can be handled. First is to turn off the query cache entirely (`query_cache_size = 0` and `query_cache_type = 0`). The second way is to manually invalidate the query cache through the use of `RESET QUERY CACHE` or similar done by an application when data is changing. This obviously would only be realistic when all data changes are occurring only through a single application and only periodically (typical during nightly batch loads in data warehouses).

Now another common cache is the MyISAM key cache (`key_buffer_size`). This is the cache that buffers indexes before being written to disk and when retrieving the data from the index. MyISAM has a mechanism built in to deal with this. Basically when using this cache with the external locking option enabled then MyISAM will detect when the underlying index file has been changed, and it will bypass the key cache and directly read the data again from the index file. This will reduce the cache hit rate, however it will ensure that the key cache does not return stale information. (An other possibility would be to disable the key cache (`key_buffer_size = 0`) but this at the costs of the performance. The the cluster file system is responsible for it.)

The final cache that we have is the table cache. This is the cache that stores the meta-data and file descriptors that MyISAM uses. This cache is not synchronized across the systems. That means that when something changes the meta data , such as DROP, ALTER or most other DDL statements that are done to an existing table, then a FLUSH TABLES command needs to be issued on all of the other nodes in order to allow them to refresh the meta data for the affected table.

Assuming these rules are followed, it should be possible to setup MyISAM in a active-active cluster. Caution is still required if one server fails however, as if it was written to the MyISAM table, then a different node will need to CHECK/REPAIR the table being written to in order to fix any of the corrupted data left behind.

One thing to note is due to the way the caches and locking works, the active-active setup is only good for performance when you have a vast majority of reads. Due to this performance issue, it typically works best for large data warehouses and very read intensive applications where data changes only rare or periodically.

It is possible tot use MyISAM in a read-only environment with multiple people accessing the same tables without all of the above settings as well.

Pitfalls and hints [7]

- Enabling external locking offers no protection against index corruption for tables that use delayed key writes.
- External locking (system locking) is disabled by default as of MySQL 4.0.
- If you use external locking on a system on which lockd does not fully work (such as Linux), it is easy for mysqld to deadlock.
- Disabling external locking does not affect MySQL's functionality as long as you run only one server.
- On some systems it is mandatory to disable external locking because it does not work, anyway.

Technical detail

According to our developers external locking is implemented by the use of “lockf”.

Literature

[1] Excerpt from the “MySQL High Availability Student Guide”, p. 5-4 ff.

[2] OCFS2: http://oss.oracle.com/projects/ocfs2/

[3] Distributed Lock Manager (DLM): http://en.wikipedia.org/wiki/Distributed_lock_manager

[4] GFS: http://en.wikipedia.org/wiki/Global_File_System

[5] GFS Project Page: http://sources.redhat.com/cluster/gfs/

[6] Comparison of file systems: http://en.wikipedia.org/wiki/Comparison_of_file_systems

[7] MySQL documentation: http://dev.mysql.com/doc/refman/5.1/en/system-optimization.html