Glock internal locking rules

This documents the basic principles of the glock state machine internals. Each glock (struct gfs2_glock in fs/gfs2/incore.h) has two main (internal) locks:

1. A spinlock (gl_lockref.lock) which protects the internal state such as gl_state, gl_target and the list of holders (gl_holders)
2. A non-blocking bit lock, GLF_LOCK, which is used to prevent other threads from making calls to the DLM, etc. at the same time. If a thread takes this lock, it must then call run_queue (usually via the workqueue) when it releases it in order to ensure any pending tasks are completed.

The gl_holders list contains all the queued lock requests (not just the holders) associated with the glock. If there are any held locks, then they will be contiguous entries at the head of the list. Locks are granted in strictly the order that they are queued, except for those marked LM_FLAG_PRIORITY which are used only during recovery, and even then only for journal locks.

There are three lock states that users of the glock layer can request, namely shared (SH), deferred (DF) and exclusive (EX). Those translate to the following DLM lock modes:

Glock mode	DLM	lock mode
UN	IV/NL	Unlocked (no DLM lock associated with glock) or NL
SH	PR	(Protected read)
DF	CW	(Concurrent write)
EX	EX	(Exclusive)

Thus DF is basically a shared mode which is incompatible with the "normal" shared lock mode, SH. In GFS2 the DF mode is used exclusively for direct I/O operations. The glocks are basically a lock plus some routines which deal with cache management. The following rules apply for the cache:

Glock mode	Cache data	Cache Metadata	Dirty Data	Dirty Metadata
UN	No	No	No	No
SH	Yes	Yes	No	No
DF	No	Yes	No	No
EX	Yes	Yes	Yes	Yes

These rules are implemented using the various glock operations which are defined for each type of glock. Not all types of glocks use all the modes. Only inode glocks use the DF mode for example.

Table of glock operations and per type constants:

Field	Purpose
go_xmote_th	Called before remote state change (e.g. to sync dirty data)
go_xmote_bh	Called after remote state change (e.g. to refill cache)
go_inval	Called if remote state change requires invalidating the cache
go_demote_ok	Returns boolean value of whether its ok to demote a glock (e.g. checks timeout, and that there is no cached data)
go_lock	Called for the first local holder of a lock
go_unlock	Called on the final local unlock of a lock
go_dump	Called to print content of object for debugfs file, or on error to dump glock to the log.
go_type	The type of the glock, `LM_TYPE_*`
go_callback	Called if the DLM sends a callback to drop this lock
go_flags	GLOF_ASPACE is set, if the glock has an address space associated with it

The minimum hold time for each lock is the time after a remote lock grant for which we ignore remote demote requests. This is in order to prevent a situation where locks are being bounced around the cluster from node to node with none of the nodes making any progress. This tends to show up most with shared mmaped files which are being written to by multiple nodes. By delaying the demotion in response to a remote callback, that gives the userspace program time to make some progress before the pages are unmapped.

There is a plan to try and remove the go_lock and go_unlock callbacks if possible, in order to try and speed up the fast path though the locking. Also, eventually we hope to make the glock "EX" mode locally shared such that any local locking will be done with the i_mutex as required rather than via the glock.

Locking rules for glock operations:

Operation	GLF_LOCK bit lock held	gl_lockref.lock spinlock held
go_xmote_th	Yes	No
go_xmote_bh	Yes	No
go_inval	Yes	No
go_demote_ok	Sometimes	Yes

Operation	GLF_LOCK bit lock held	gl_lockref.lock spinlock held
go_lock	Yes	No
go_unlock	Yes	No
go_dump	Sometimes	Yes
go_callback	Sometimes (N/A)	Yes

> **Note**
>
> Operations must not drop either the bit lock or the spinlock if its held on entry. go_dump and do_demote_ok must never block. Note that go_dump will only be called if the glock's state indicates that it is caching uptodate data.

Glock locking order within GFS2:

1. i_rwsem (if required)
2. Rename glock (for rename only)
3. Inode glock(s) (Parents before children, inodes at "same level" with same parent in lock number order)
4. Rgrp glock(s) (for (de)allocation operations)
5. Transaction glock (via gfs2_trans_begin) for non-read operations
6. i_rw_mutex (if required)
7. Page lock (always last, very important!)

There are two glocks per inode. One deals with access to the inode itself (locking order as above), and the other, known as the iopen glock is used in conjunction with the i_nlink field in the inode to determine the lifetime of the inode in question. Locking of inodes is on a per-inode basis. Locking of rgrps is on a per rgrp basis. In general we prefer to lock local locks prior to cluster locks.

Glock Statistics

The stats are divided into two sets: those relating to the super block and those relating to an individual glock. The super block stats are done on a per cpu basis in order to try and reduce the overhead of gathering them. They are also further divided by glock type. All timings are in nanoseconds.

In the case of both the super block and glock statistics, the same information is gathered in each case. The super block timing statistics are used to provide default values for the glock timing statistics, so that newly created glocks should have, as far as possible, a sensible starting point. The per-glock counters are initialised to zero when the glock is created. The per-glock statistics are lost when the glock is ejected from memory.

The statistics are divided into three pairs of mean and variance, plus two counters. The mean/variance pairs are smoothed exponential estimates and the algorithm used is one which will be very familiar to those used to calculation of round trip times in network code. See "TCP/IP Illustrated, Volume 1", W. Richard Stevens, sect 21.3, "Round-Trip Time Measurement", p. 299 and onwards. Also, Volume 2, Sect. 25.10, p. 838 and onwards. Unlike the TCP/IP Illustrated case, the mean and variance are not scaled, but are in units of integer nanoseconds.

The three pairs of mean/variance measure the following things:

1. DLM lock time (non-blocking requests)
2. DLM lock time (blocking requests)
3. Inter-request time (again to the DLM)

A non-blocking request is one which will complete right away, whatever the state of the DLM lock in question. That currently means any requests when (a) the current state of the lock is exclusive, i.e. a lock demotion (b) the requested state is either null or unlocked (again, a demotion) or (c) the "try lock" flag is set. A blocking request covers all the other lock requests.

There are two counters. The first is there primarily to show how many lock requests have been made, and thus how much data has gone into the mean/variance calculations. The other counter is counting queuing of holders at the top layer of the glock code. Hopefully that number will be a lot larger than the number of dlm lock requests issued.

So why gather these statistics? There are several reasons we'd like to get a better idea of these timings:

1. To be able to better set the glock "min hold time"
2. To spot performance issues more easily
3. To improve the algorithm for selecting resource groups for allocation (to base it on lock wait time, rather than blindly using a "try lock")

Due to the smoothing action of the updates, a step change in some input quantity being sampled will only fully be taken into account after 8 samples (or 4 for the variance) and this needs to be carefully considered when interpreting the results.

Knowing both the time it takes a lock request to complete and the average time between lock requests for a glock means we can compute the total percentage of the time for which the node is able to use a glock vs. time that the rest of the cluster has its share. That will be very useful when setting the lock min hold time.

Great care has been taken to ensure that we measure exactly the quantities that we want, as accurately as possible. There are always inaccuracies in any measuring system, but I hope this is as accurate as we can reasonably make it.

Per sb stats can be found here:

```
/sys/kernel/debug/gfs2/<fsname>/sbstats
```

Per glock stats can be found here:

```
/sys/kernel/debug/gfs2/<fsname>/glstats
```

Assuming that debugfs is mounted on /sys/kernel/debug and also that <fsname> is replaced with the name of the gfs2 filesystem in question.

The abbreviations used in the output as are follows:

srtt	Smoothed round trip time for non blocking dlm requests
srttvar	Variance estimate for srtt
srttb	Smoothed round trip time for (potentially) blocking dlm requests
srttvarb	Variance estimate for srttb
sirt	Smoothed inter request time (for dlm requests)
sirtvar	Variance estimate for sirt
dlm	Number of dlm requests made (dcnt in glstats file)
queue	Number of glock requests queued (qcnt in glstats file)

The sbstats file contains a set of these stats for each glock type (so 8 lines for each type) and for each cpu (one column per cpu). The glstats file contains a set of these stats for each glock in a similar format to the glocks file, but using the format mean/variance for each of the timing stats.

The gfs2_glock_lock_time tracepoint prints out the current values of the stats for the glock in question, along with some addition information on each dlm reply that is received:

status	The status of the dlm request
flags	The dlm request flags
tdiff	The time taken by this specific request

(remaining fields as per above list)