Device Locks: Mutual Exclusion for Storage Area Networks

Kenneth W. Preslan, Steven R. Soltis, Christopher J. Sabol, and Matthew T. O'Keefe
Parallel Computer Systems Laboratory
University of Minnesota

Gerry Houlder and Jim Coomes
Seagate Technology

Abstract

Device Locks are mechanisms used in distributed environments to facilitate mutual exclusion of shared resources. They can further be used to maintain coherence of data that is cached in several locations. The locks are implemented on the storage devices and accessed with the SCSI device lock command, *Dlock*.

This paper presents the Dlock command and discusses how it can be used as a mutual exclusion device for Storage Area Networks and Shared Disk File Systems. Methods for the recovery of a Dlock held by a failed initiator are also presented.

The Dlock command is in the process of being standardized as part of the SCSI 3 specification.

1 Introduction

Device locks are advisory locks implemented on a SCSI device that allow the mutual exclusion of devices accessing a shared resource. The locks are manipulated using the Dlock SCSI command. The Dlock command is independent of all other SCSI commands, so devices supporting the locks have no awareness of the nature of the resource that is locked. Each lock requires only a small amount of memory allowing devices to support thousands of these locks with minimal amounts of memory.

Each Dlock can be acquired in one of two modes, shared or exclusive. A shared lock allows multiple clients to access the data protected by the lock as long as they don't change it. This allows multiple readers of a directory or file at the same time. If a client needs to change the data, it acquires the lock in exclusive mode. No one else can read or write the data for the duration that the lock is held.

Dlocks can also timeout after a period of inactivity. If a lock is acquired and it is not unlocked in a certain amount of time, the lock automatically changes to an unlocked state. The next initiator that tries to get the lock will succeed and be notified that the previous holder let the lock expire. The actual length of the timeout can be set in the Dlock Mode Page. This timeout, together with the Report Expired action (which reports which locks have timed out), facilitates a "cleaner" program that runs in the background looking for and fixing the messes left by client failures.

The Refresh Lock action can be used to reset the timer on the lock. This allows a client to hold a lock for longer than one timeout period, but also allows the lock to timeout if the client fails.

Each device lock has a state field, an activity bit, and a version number. After all Dlock operations, the current values are returned to the initiator where the values are saved for measurement of lock activity. Activity measurements are useful in the event of failures, for load balancing shared resources, and for maintaining data coherence.

Device locks support six primary actions: Lock Shared, Lock Exclusive, Unlock, Unlock Increment, Force Lock Exclusive, and Refresh Lock. The actions use test-and-set and clear operations to modify the lock state. The version number is incremented according to the actions and activity bit. The Force Lock action takes a version number as an input to be compared with the current value to determine if the lock should be taken away from its current holder and set in the Locked Exclusive state under ownership of the new initiator.

Device locks support four secondary actions: No Action, Activity On, Activity Off, and Report Expired. These actions do not modify the lock state. The No Action action may be used to read the state, activity, version number, and other information about the lock. The Activity On and Activity Off actions set and clear the activity bit, respectively. The Report Expired action returns a bitmap that tells the initiator which locks have expired.

The version numbers are incremented after successful Unlock Increment, Force Lock Exclusive, and Activity Off actions. The version numbers are also incremented after successful Unlock actions provided the activity bit is set. The size of the version number is 32 bits. The initiators must be aware that the version number periodically rolls-over from its maximum value to zero. The minimum roll-over time can be determined by time stamping the access of each lock. If the current time differs from the last access time by some amount less than the known roll-over time, then the clock is guaranteed not to have rolled. The client can use these version numbers to implement cache coherency schemes.

An important concept in Dlock is the Client ID Number. A Client ID number is a arbitrary 32-bit number that identifies a client to other clients. Client IDs are an opaque value to Dlock devices. In other words, the device needs to remember the Client ID of the initiator holding a Dlock, but it doesn't need to understand its contents.

Every Dlock Lock Action that is passed to a Dlock device includes the Client ID of the initiator sending the command. The device stores the Client ID and returns it to other initiators accessing the lock as a way of communicating the name of initiator that the lock. These other initiators can then use the Client ID to implement sophisticated failure recovery schemes.

The Dlock command is in the process of being standardized as part of the SCSI 3 specification [1], [2].

2 The Dlock Interface

The Dlock interface is defined by three main parts, the CDB, the return data, and the mode page.

2.1 Dlock CDB

The Dlock CDB is show in Table 1. The fields of interest are:

Operation Code The SCSI Operation code for Dlock. This is currently A0h, but will very likely change in the future.

Action This describes the action being requested. The possible values of this field are shown in Table 2.

Lock Number This is the number of the lock to be operated on.

Client ID A application-defined 32 number that identifies the client issuing the Dlock command.

Input Version Number One of the actions, Force Lock Exclusive, only completes successfully if the least significant byte of the lock's version number is equal to this field.

Allocation Length The number of bytes that the initiator has allocated for data returned from the command.

2.2 Dlock Return Data

The format of the data returned by the Dlock command depends on what action was issued.

2.2.1 Type 1 Return Data

Type 1 return data is returned from all actions except Report Expired. The target will return information about the state of the lock operated on by the CDB. The information returned represents the state of the lock after the current command was completed. The exception to this is the Expired Field. The Expired Field indicates the state of the lock before the command, if the action was Lock Shared, Lock Exclusive, or Force Lock Exclusive. The format of the data is shown in Table 3.

Return Data Length The length in bytes of the returned data.

Result This bit is 1 if the action succeeded, 0 if the action failed.

Activity This bit is 1 if activity monitoring is on, 0 if it is off.

Expired This field indicates whether or not the initiator that last held this lock released it cleanly or let it time out. The values of this field are shown in Table 4

State The values of the State of the lock are shown in Table 5.

Version Number This is the version number of the lock.

Number of Holders This is the number of clients currently holding this lock.

Client ID Numbers This is a list of the Client ID Numbers of all the initiators currently holding this Dlock.

2.2.2 Type 2 Return Data

Type 2 data is returned from the Report Expired action. The format of the data returned is shown in Table 6. Each bit in the bitmap is a one if the associated lock has expired.

Byte,Bit	7	6	5	4	3	2	1	0
0	Operation Code (83h)							
1	Reserved				Action			
2	(MSB)	Lock Number						
3								
4								
5								(LSB)
6	(MSB)	Client ID						
7								
8								
9								(LSB)
10	(MSB)	Allocation Length						
11								
12								
13								(LSB)
14	Input Version Number (LSB)							
15	Control							

Table 1: Dlock CDB

Code	Action	Description	Data Returned
0h	Nop	No change, return current lock state	Type 1
1h	Lock Shared	Acquire shared lock	Type 1
2h	Lock Exclusive	Acquire exclusive lock	Type 1
3h	Force Lock Exclusive	Acquire exclusive lock, preempting lock if needed	Type 1
4h	Refresh Lock	Reset timer on lock	Type 1
5h	Unlock	Release lock	Type 1
6h	Unlock Increment	Release lock and increment version number	Type 1
7h	Activity On	Turn on Activity Monitor	Type 1
8h	Activity Off	Turn off Activity Monitor	Type 1
9h	Report Expired	Report which locks have expired	Type 2
Ah–Fh	Reserved	Reserved	

Table 2: Dlock Actions

Byte,Bit	7	6	5	4	3	2	1	0
0	(MSB)			Version Number				
1								
2								
3								(LSB)
4	Result	Act	Reserved		Expired		State	
5				Number of Holders				
6	(MSB)			Holder List Length ($n-7$)				
7								(LSB)
				List of Holders				
8	(MSB)			Client ID (first)				
11								(LSB)
				:				
				:				
$n-3$	(MSB)			Client ID (last)				
n								(LSB)

Table 3: Type 1 Reply Data Format

Code	Description
0h	Not Expired
1h	Expired from Locked Shared
2h	Expired from Locked Exclusive
3h	Reserved

Table 4: Values of the Expired field

Code	Description
0h	Unlocked
1h	Locked Shared
2h	Locked Exclusive
3h	Reserved

Table 5: Values of the State field

Byte,Bit	7	6	5	4	3	2	1	0
0	Result	Reserved						
1	Reserved							
2	(MSB)	Data Length ($n - 3$)						
3								(LSB)
	Bitmap of expired device locks							
2	L7	L6	L5	L4	L3	L2	L1	L0
3	L15	L14	L13	L12	L11	L10	L9	L8
4	L23	L22	L21	L20	L19	L18	L17	L16
	:							
n	Lm							
	Where $m = (n(- 3) * 8 - 1)$							

Table 6: Type 2 Reply Data Format

Byte,Bit	7	6	5	4	3	2	1	0
0	PS	Resvd	Page Code (xxh)					
1	Page Length (0Ah)							
2	Reserved							
3	Maximum clients per lock							
4	(MSB)	Number of locks						
5								
6								
7								(LSB)
8	(MSB)	Lock Timeout Interval (ms)						
9								
10								
11								(LSB)

Table 7: Mode Page Data

2.3 Mode Page

The mode page returns information about lock parameters.

Maximum number of clients able to share a lock This is the number of clients that can simultaneously hold a lock in the Locked Shared state.

Number of locks on the device Returns the number of Dlocks on the device.

Lock Timeout Interval The number of milliseconds after which an refreshed lock will timeout. If this value is zero, locks never time out.

3 Dlock Behavior

There are a number of details of the Dlock specification that aren't obvious from the interface.

3.1 Actions

A more detailed description of the actions can be seen in Table 8. Important things to notice are:

- If a lock has expired from a Locked Exclusive state, a Lock Shared request should produce a Locked Exclusive state. This allows the client acquiring the lock to do whatever cleanup is necessary without exposing other clients to data in an inconsistent state.
- If a client holds a lock as Locked Exclusive, it should be able to issue a Lock Shared command on the same lock and the lock will be converted to the Locked Shared state.
- If a client holds a lock as Locked Shared and it is the only client holding the lock at the time, it should be able to issue a Lock Exclusive command on the same lock and the lock will be converted to the Locked Exclusive state. If the lock is shared by other initiators, the Exclusive Pending bit (see Section 3.4) is set.
- A client may hold multiple instances of the same lock, if it is in a Locked Shared state. This is a valid (if somewhat useless) condition.
- If a Force Lock Exclusive is issued, the lock is held, and the low three bytes of the input version number is correct, the FLE's return data should indicate that the lock expired. Example – If a lock is held by one client in a Lock Shared state and another client issues a Force Lock Exclusive, the Expired field in the return data should be "Expired from Lock Shared". This allows a client to know if recovery is required. If a Force Lock Exclusive is issued on an unlocked lock, the Expired field should be "Not Expired".
- If a Refresh Lock action with a lock number of all Fs is issued, the target resets the timer on all the locks currently held by that initiator.

3.2 Version Numbers and Caching

The main reason to have version numbers associated with each lock is to provide a means of determining the consistency of a client's local data cache. Before accessing (either reading or writing) any data, the appropriate device lock is acquired. When finished the lock is released. An Unlock is used when the data has not been modified during the operation. Otherwise an Unlock Increment must be used to signify that the data was modified.

This data can be cached in a system's memory, though its consistency is unknown until the next successful Lock action. The consistency of cached data is determined based on the version number of the lock. The data is consistent if the version number returned by the Lock operation is the same as what was returned from the initiator's previous Unlock or Unlock Increment action. Otherwise, the data must be reread. Modified data can be cached while the Dlock is held, but must be written though to the device before the Dlock is released. This criteria assumes that roll-over of the version number has not occurred. Given roll-over, the cache must be treated as inconsistent.

Table 9 shows example accesses from two initiators, A and B. Each event is ordered based on the time given in the left most column. The Action column represents a device lock command sent to the device assuming all commands access the same lock. The State field gives the lock state (U=Unlocked, S=Locked Shared, and E=Locked Exclusive) and the version number value. The activity monitor bit is always 0. Version numbers of x represent values that are assumed to have rolled. The state field is updated by the return of each command and does not change when the other initiator performs a command. The Consistent field states whether or not the cached data is consistent. Finally, the Lock field gives the current state of the device lock.

The first two Locks from each initiator are not consistent, because the version number has been assumed to have rolled-over. The Lock at time 7 guarantees that the data is consistent, since the new version number is equal to the old version number. An Unlock Increment is used at

	State		
Action	Unlocked	Locked Shared	Locked Exclusive
Nop	Return.result ← 1	Return.result ← 1	Return.result ← 1

Lock Shared	Return.result ← 1
	if Lock[N].expired = ExpiredExclusive
	    Lock[N].state ← LockedExclusive
	else
	    Lock[N].state ← LockedShared
	Lock[N].holders ← 1
	Add ClientID to the list
	Reset expire time
		if Lock[N].holders < MaxHolders
		    Return.result ← 1
		    Lock[N].holders++
		    Add ClientID to the list
		    Reset expire time
		else
		    Return.result ← 0
			if Lock[N].ClientID[0] = ClientID
			    Return.result ← 1
			    Lock[N].state ← LockedShared
			    Reset expire time
			else
			    Return.result ← 0

Lock Exclusive	Return.result ← 1
	Lock[N].state ← LockedExclusive
	Lock[N].holders ← 1
	Add ClientID to the list
	Reset expire time
		if (Lock[N].holders = 1 and
		        Lock[N].ClientID[0] = ClientID)
		    Return.result ← 1
		    Lock[N].state ← LockedExclusive
		    Reset expire time
		else
		    Return.result ← 0
			Return.result ← 0

Force Lock Exclusive	Return.result ← 1
	Lock[N].state ← LockedExclusive
	Lock[N].holders ← 1
	Add ClientID to the list
	Reset expire time
		if Lock[N].version = version
		    Return.result ← 1
		    Lock[N].state ← LockedExclusive
		    Lock[N].holders ← 1
		    Lock[N].version++
		    Clear list, then Add ClientID
		    Reset expire time
		else
		    Return.result ← 0	

Refresh Lock	Return.result ← 0	if *ClientID is in the list*
		    Return.result ← 1
		    if *Lock Number is 0xFFFFFFFF*
		        Reset expire time on all locks held by this initiator
		    else
		        Reset expire time
		else
		    Return.result ← 0	

Unlock	Return.result ← 0	if *ClientID is in the list*
		    Return.result ← 1
		    Lock[N].holders−−
		    Remove ClientID from the list
		    if Lock[N].activity = 1
		        Lock[N].version++
		    if Lock[N].holders = 0
		        Lock[N].state ← Unlocked
		else
		    Return.result ← 0	

Unlock Increment	Return.result ← 0	if *ClientID is in the list*
		    Return.result ← 1
		    Lock[N].holders−−
		    Remove ClientID from the list
		    Lock[N].version++
		    if Lock[N].holders = 0
		        Lock[N].state ← Unlocked
		else
		    Return.result ← 0	

Activity On	Return.result ← 1
	Lock[N].activity ← 1		

Activity Off	Return.result ← 1
	Lock[N].activity ← 0
	Lock[N].version++		

All Actions	Return.activity ← Lock[N].activity
	Return.expired ← Lock[N].expired
	Return.state ← Lock[N].state
	Return.version ← Lock[N].version
	Return.ClientID[] ← Lock[N].ClientID[]		

Table 8: Device Lock Actions

	Initiator A			**Lock**	**Initiator B**		
	Action	State	Consistent		Action	State	Consistent
0		U,x		U,0		U,x	
1	Lock Shared	S,0	No	S,0		U,x	
2	No Modify	S,0		S,0		U,x	
3	Unlock	U,0		U,0		U,x	
4		U,0		S,0	Lock Shared	S,0	No
5		U,0		S,0	No Modify	S,0	
6		U,0		U,0	Unlock	U,0	
7		U,0		E,0	Lock Exclusive	E,0	Yes
8		U,0		E,0	Modify	E,0	
9		U,0		U,1	Unlock Incr	U,1	
10	Lock Shared	S,1	No	S,1		U,1	
11	Modify	S,1		S,1		U,1	
12	Unlock Incr	U,2		U,2		U,1	
13		U,2		S,2	Lock Shared	S,2	No
14		U,2		S,2	No Modify	S,2	
15		U,2		U,2	Unlock	U,2	
16	Lock Exclusive	E,2	Yes	E,2		U,2	
17	No Modify	E,2		E,2		U,2	
18	Unlock	U,2		U,2		U,2	

Table 9: Device Lock Example

time 9 to signify that data was modified. The Lock at time 10 makes no guarantee that the data is consistent, since in this case the version numbers differ. The next Lock again assumes the cache to be inconsistent on the basis that the version numbers differ. The final Lock shows the data is consistent, since the version number has not changed since time 12.

In general, Locked Shared actions will not be followed by Unlock Increment actions. Lock Shared implies a read of the data, without a modify, so there is no need to invalidate the caches of the other initiators. There is no reason to prohibit a Locked Shared/Unlock Increment pair, though. Similarly, Locked Exclusive can be followed by either Unlock or Unlock Increment.

3.3 Handling Client Failures

Much of the work that has been put into the Dlock specification deals with recovering from initiator failures. Without these failure detection schemes, an initiator that fails could leave Dlocks in locked states indefinitely. This would be bad.

The general Dlock philosophy has always been to try to avoid having to make clients aware of each other. This means that clients should be able to detect the failures of other clients solely by interacting with the Dlock device. There are currently two methods of doing this, Activity and Dlock Timeouts.

There is also a method of detecting failures when the clients talk to each other directly. Since Dlock commands return a list of the Client ID Numbers of the hosts holding a lock, it is possible to implement failure detection at a higher level. The ability of these three failure detection schemes offers a great deal of flexibility to systems that use Dlock.

3.3.1 Activity/Force Lock Exclusive

An initiator attempting to acquire a lock that is owned by a failed initiator can identify that the lock has not been accessed by checking the activity of the lock's version number. A changing version number indicates that the lock is actively being used. A unchanging version number is a symptom of a failed client. The number will also not change if a the lock is continually being acquired and released with an Unlock command.

If no lock activity is observed, the initiator turns on the activity monitoring by the Activity On actions. The version number now is incremented for both Unlock and Unlock Increment operations. If the version number shows activity, the initiator turns off the activity monitoring and attempts to acquire the lock knowing it is not held by a failed initiator. If the version number remains unchanged, the initiator performs a Force Lock Exclusive on the lock.

The Force Lock Exclusive action ensures that the lock will only be reset by the initiator that can identify the value of the current version number. This solves the the case where two separate initiators simultaneously attempt to reset the same lock. The first initiator forces the lock, which increments the version number. The second initiator's Force action is ignored.

Only the three least significant bytes are passed in to the device in the Force Lock Exclusive action. This is sufficient because a Force Lock Exclusive action is always preceded by attempts at acquiring the lock using a lock action. The return data from these attempts provides the initiator with the current version number. The initiator then uses that value in the Force Lock Exclusive. The only way the version number can change between these two actions is if it is incremented by another initiator's Force Lock Exclusive or Unlock Increment. The low three bytes of the version number are sufficient to determine this.

When an initiator eventually succeeds with the Force Lock Exclusive, it is told what state the lock was in. It can then perform recovery on the lock's data. Recovery from a Locked Shared state may be very different from recovery from a Locked Exclusive state.

3.3.2 Dlock timeouts/Refresh Lock/Report Expired

The new method of detecting failed clients employs Dlock timeouts. Clients don't have to keep track of which other clients have failed because the Dlock device does it for them. If a initiator ever leaves a Dlock locked for more than a timeout period, the Dlock device assumes the client has failed and unlocks the lock.

When the lock times out, the device sets a expired flag on the lock. The flag contains information on whether the lock was held in shared or exclusive state. When a new client tries to acquire the lock, it succeeds because the lock was reset. It can then examine the expired flag and perform the appropriate recovery operation.

An initiator can prevent a Dlock from timing out by using the Refresh Lock action. This action resets the timer on the lock. A client who acquires a lock and performs a refresh lock action on it periodically to prevent timeout can hold that lock indefinitely.

The Dlock mode page contains a field for setting the Dlock timeout interval. This value can be set to zero, which disables Dlock timeout. This allows even greater flexibility.

The Report Expired action returns a bitmap of all the locks. If a bit is one, the corresponding Dlock has expired. This mechanism allows a dedicated program that runs in the background on the initiator to check for expired locks and recover them. The program doesn't have to go linearly though the locks, acquiring and unlocking them. One Report Expired action allows the program to get a list of all the expired locks.

A "cleaner" program like this is desirable because it can limit the maximum amount of time a file system can be in an inconsistent state after a client failure. If the program does a Report Expired once a minute, the cleaner will know about an expired lock in less than a minute. Without the cleaner, a client failure on a infrequently used lock could go undetected for a long time.

3.3.3 Client ID Numbers

Dlock commands always return a list of Client ID Numbers of the initiators holding the Dlock. This allows an arbitrary failure detection scheme to be implemented. Once a client has the Client ID of the client holding a lock, a non-SCSI failure detection protocol can be used. The client can determine for itself whether or not another client is down. It can then use the Force Lock Exclusive action to reset the lock, if necessary.

3.4 Shared and Exclusive Locks

As stated before, a Dlock can be acquired in one of two modes, Shared or Exclusive. A number of clients can hold the same lock at the same time if it's in a shared state. This allows many different clients to read the same piece of data at the same time. If a client wants to write, it acquires an Exclusive lock. This allows a multiple readers/single write scheme to be implemented.

The problem with this scenario is that a progression of readers can cause a writer to wait indefinitely. Two or more clients can hold a lock in a shared state forever, even if they are constantly releasing and re-acquiring the lock. The first client gets the lock, the second client gets the lock, the first client releases the lock, the first client re-acquires the lock, the second client releases the lock, and so on. There is no time when the lock is unlocked. The lock is constantly being locked and unlocked, so it doesn't time out. A client that wants to get the lock exclusively waits forever.

The solution to this problem is to implement an "exclusive pending" bit. If a lock is held in the shared state and a Lock Exclusive action comes in, the exclusive pending bit is set. This means that all Locked Shared actions will fail until the lock switches to an unlocked state. All the current readers drain off until the writer gets a chance at the lock.

There is one problematic detail. Imagine that there are a bunch of initiators sharing a lock. A Lock Exclusive action then comes in and fails, which sets the Exclusive Pending bit. No new Lock Shared requests succeed and eventually the count reaches zero. In a perfect world, the client wanting the lock exclusively then gets the lock and all clients go merrily on their way.

In an imperfect world, another client will try to do a Lock Shared before the client that wants the exclusive lock can retry. At this point, there are three different strategies that can be implemented:

1. The Dlock device forgets that a Lock Exclusive is pending and clears the bit. The Lock Shared succeeds and the client that wants the lock exclusively has to wait for all of the shared clients to drain off again.

2. All Lock Shared actions are rejected until a Lock Exclusive succeeds. When a Lock Exclusive action succeeds the Exclusive Pending bit is cleared. This is bad because the client trying to do the Lock Exclusive might have failed or changed its mind. It doesn't get the lock exclusively and all the "Lock Shared" clients have to wait until some other random client decides it wants the lock exclusively.

3. Allow one Lock Shared action to succeed, but keep the Exclusive Pending bit on. Clients requesting a shared lock are allowed to get it, but only one client at a time. This keeps "Lock Shared" clients from starving, but provides the maximum chance for the "Lock Exclusive" client to complete its work. Once a Lock Exclusive succeeds, the Exclusive Pending bit is cleared and multiple Locked Shared actions can succeed again.

This specification proposes that option number three be implemented. This allows the maximum fairness, with the minimum slowdown in case of a failure. A state transition diagram for this algorithm is shown in Figure 1.

3.5 Client IDs and GFS

The new Global File System [3] architecture will take advantage of the fact that Dlock commands return Client ID Numbers. Clients will hold Dlocks for long periods of time (using Refresh Lock to prevent timeout). While the client holds the Dlock, it can do write caching of the data protected by that Dlock. This lets GFS relax its synchronous nature and improve write performance.

When a second client needs to use the lock, it attempts to acquire the lock, but fails. In the return data from the failed lock action, the Client ID of the Dlock holder is returned. The second client can then map this Client ID into a network address. It can then make a request to the first client, asking it to release the Dlock. The first client then writes back its cached data and releases the lock.

Later versions of GFS can be even more aggressive and use this out-of-band communication to do things like Third–Party Transfer. The client holding a Dlock doesn't even have to write back its cache to allow other clients access to the same data. If the second client is requesting something that is not in the first client's cache, the first client can just give permission to the second client permission to read the data from disk. If the data requested is small enough and in the first client's cache, the first client could even forward the data directly to the second client.

3.6 Dlock Persistence

Dlocks are intended to be light weight locks that provide a quick response time. Having a hold on a Dlock that persist across the power cycle of a device would be nice, but it's not necessary. Lock states do not need to be stored in non-volatile memory. Specific conditions and how they should be handled are shown in Table 10.

If an initiator is logged out of a device because there are too many other devices, it should not lose its Dlocks. When the initiator next logs in to the device it should be able to perform all the actions it could if it didn't log out. This doesn't affect the timeout, however. If the timer on a Dlock expires when the initiator that holds it isn't logged in, the Dlock is put in the expired state, just as it would if the initiator was logged in.

4 Implementation Details

There are a number of details to the Dlock concept that can improve its efficiency without changing the interface. Some of these are:

4.1 Out-of-Queue Dlocks

One of the things that hurts the performance of GFS is the fact that Dlock operations can be stuck behind large data transfers in a drive's command queue. The client has to wait for unrelated disk operations to happen before it can continue.

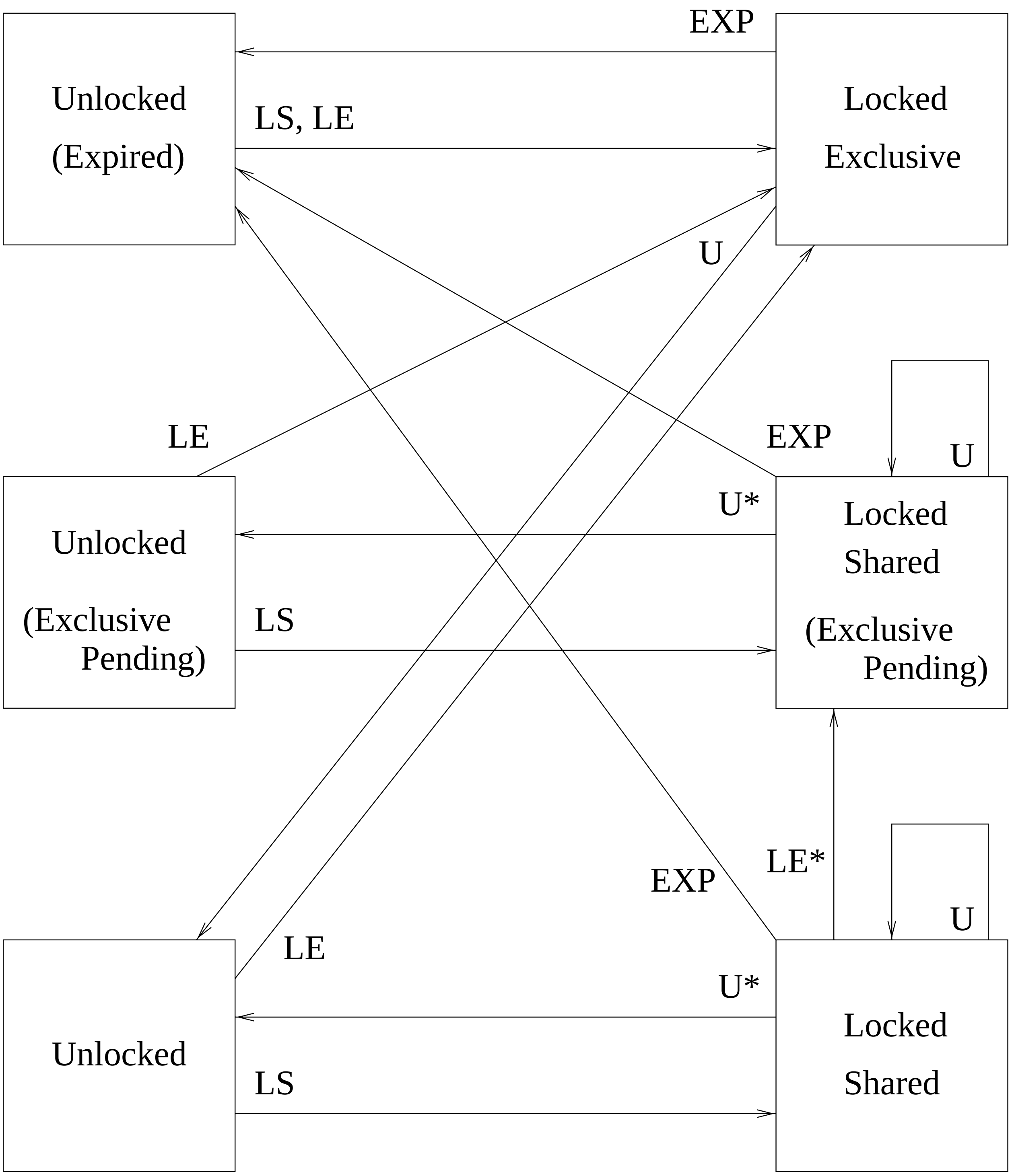

Figure 1: A state transition table that takes into account the Exclusive Pending bit. State transitions are described by these abbreviations: EXP=Lock Expiration, LS=Lock Shared Action, LE=Lock Exclusive Action, LE*=Lock Exclusive Action from another initiator, U=Unlock Action, U*=Last Unlock Action on a shared lock

Condition	Condition handling
Mode Select on Dlock Dlock Mode Page	Issue Unit Attention: *All Device Locks Cleared* for all initiators All lock values are zeroed
Power Cycle	Issue Unit Attention: *Power On* for all initiators All lock values are zeroed
SCSI Reset *	Issue Unit Attention: *Power On* for all initiators All lock values are zeroed
Bus Device Reset *	Lock values not affected
Task Management ** Target Reset	Lock values not affected
Initiator Logout **	Locks held by that initiator remain held.

Table 10: Dlock Power Cycles, Resets, and Mode Selects: * Parallel SCSI only, ** Fibre Channel only

The Dlock command has low processor overhead. There are no disks accesses involved. There are no large calculations involved. The Dlock SCSI commands could be implemented outside of the command queue. Dlock commands could be acted on as they are pulled off the wire. This may slightly increase the devices's firmware complexity, but it would be a big win performance–wise.

4.2 Event–driven Timeouts

A Dlock device can detect Dlock timeouts in one of two ways:

1. A thread in the Dlock device could wake up periodically and check to see if any of the locks have expired. The thread would scan linearly though all the locks checking the time stamps and marking the expired locks as unlocked and expired. If there were a large number of Dlocks on a device, this could take a fair amount of process time for each scan. Scans would have to be frequent, so a lot of time would be wasted.

2. Make timeout checking event driven. Only check the time stamps on a lock when you get a request for that lock. This distributes the workload so that it doesn't happen all at once. It also means that the device doesn't have to waste time checking locks that aren't being used. This is a much better approach.

4.3 Client ID Numbers

Storing a list of Client IDs for each Dlock can use up too much memory. Just storing an offset into a device wide list of Client IDs is enough to reconstruct the Client ID list for the Type 1 Return Data Block. This saves memory. Care must be taken, though, so that the list is still valid if a initiator gets logged off of the device because there are too many others logged in.

5 Future Work – Multiple Actions Per Command

In the current Dlock Specification, all deadlock detection and avoidance must be done by the initiators. If a client needs to hold two Dlocks, it must get them one at a time. If another client wants the same two Dlocks, but gets them in the opposite order, deadlock can result.

GFS handles this problem by implementing a complicated system of back-offs and retries. If a client is holding one Dlock and wants another, it tries to get the new lock for a certain amount of time. If it doesn't get the lock in this time, it assumes a deadlock condition exists. It releases the first lock, sleeps for a random amount of time, and then retries the whole operation. This system solves the deadlock problem, but it is not fun to implement and it is not time optimal.

A better solution would be to this problem would be to implement a Dlock command that allowed multiple actions. In the above situation, an initiator would issue a command that contained two lock actions. The command would be atomic so that: **1)** Either both actions would succeed and the initiator would now hold both locks, or **2)** both actions would fail and the client would hold neither of the locks. This eliminates the deadlock problem.

The Multiple Action idea could be expanded to allow many unrelated Dlock actions to be issued in one atomic Dlock command. This could be a very powerful tool. There are problems with implementing this scheme, though.

Packing multiple actions into one command requires more space than is currently available in a 12 byte CDB. The current possible work-arounds are:

1. Implement one Command with both a data-in and a data-out phase. This is possible in Fibre Channel, but frowned upon in Parallel SCSI. Using parallel SCSI as a testbed is a very nice thing, so we would prefer not to implement this option.

2. Implement two separate commands, one with a data-in phase and one with a data-out phase. This approach would cause synchronization problems. How would a device match the "Issue Action" command with the "Request Result" command? Also, this technique forces the initiator to issue two commands per lock request. Dlock is intended to be a lightweight command and this doubles the overhead of the lock/unlock process.

In the future, a third option would allow the advantages of multiple actions per command, but at the current time it is more trouble that it is worth.

6 Acknowledgments

Many people have contributed code and ideas to The Dlock Specification over the years. The people we can remember at the moment are:

- From the **University of Minnesota**
  Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Benjamin I. Gribstad, Erling Nygaard, Thomas M. Ruwart, Aaron Sawdey, David C. Teigland

- From **Seagate Technology, Inc.**
  Dave Anderson, Tony Hecker, Nate Larson, Michael H. Miller, Troy Wheeler

- From **NASA Ames Research Center**
  Alan Poston, John Lekashman

- From **Ciprico, Inc.**
  Raymond Gilson, Edward A. Soltis

References

[1] Matthew T. O'Keefe, Kenneth W. Preslan, Christopher J. Sabol, and Steven R. Soltis. X3T10 SCSI committee document T10/98-225R0 – Proposed SCSI Device Locks. http:// ftp.symbios.com/ ftp/ pub/ standards/ io/ x3t10/ document.98/ 98-225r0.pdf, September 1998.

[2] X3T10 SCSI committee. Document T10/98-225R1 – Proposed SCSI Device Locks. http:// ftp.symbios.com/ ftp/ pub/ standards/ io/ x3t10/ document.98/ 98-225r1.pdf, October 1998.

[3] Preslan et al. A 64-bit, shared disk file system for linux. In *The Sixteenth IEEE Mass Storage Systems Symposium held jointly with the Seventh NASA Goddard Conference on Mass Storage Systems & Technologies*, San Diego, California, March 1999.