SecureShare™[1]

Safe UNIX/Windows File Sharing through Multiprotocol Locking

[1] Patent Pending

Andrea Borr

Agenda

Multiprotocol File Sharing in Mixed UNIX/Windows Networks

Problems with Uncoordinated Concurrent Reads and Writes

Locking Model of CIFS

Locking Model of UNIX with NFS/NLM

Issues Impeding CIFS, NFS/NLM Interoperability

Features of SecureShare = Multiprotocol Lock Manager

The Uniform Lock-Mode Model

Coping with NFS/NLM's Lack of File-Open

The CIFS Oplock Model

Why NFS and NLM Must Break Oplocks

Example: NFS **rm** Encounters an Oplocked File...

Conclusions

Multiprotocol File Sharing in Mixed UNIX/Windows Networks

UNIX Clients -- **NFS** or **NFS**/**NLM**

Windows Clients -- **CIFS** or **(PC)NFS**

- Concurrent requests on shared server **files**, **directories:**
  - NFS **read, write, create**, **rm**, **rmdir**, **mv**, etc.
  - NLM **byte-range lock**
  - CIFS **open**, **read**, **write**, **close**, **create**, **delete**, **rename**, **move**, etc.
  - CIFS **byte-range lock**

Problems with Uncoordinated Concurrent Reads and Writes

Application Failures

File Data Integrity Problems

Cache Coherence Problems

Examples of Problems:

1 Readers receive stale data (currently being updated by another application)

2 Writers overwrite each others' updates

3 Applications have in-use files deleted or renamed “out from under” them

Locking Model of CIFS

CIFS Avoids Problems 1-3 by Assuming that the Server & all of Clients Conform to:

- Hierarchical Locking
  - Application must **open** file (getting file-lock) before doing **reads**, **writes**, **byte-range locks**, ..
  - Open specifies *access-mode* for requester (Read, Write, Read-Write) and *deny-mode* for others (Deny-None, Deny-Read, Deny-Write, Deny-All)
- Mandatory Locking
  - System validates **reads**, **writes** against file-locks, byte-range locks
  - Disallows read/write of file except under an open with appropriate *access-mode*
  - Disallows write/read of byte-range (non)exclusive-locked by another
  - Disallows **open** with *access-mode* incompatible with previous open's *deny-mode* or a *deny-mode* incompatible with previous open's *access-mode*

Locking Model of UNIX with NFS/NLM

- Non-hierarchical Locking, Lack of File-Open
  - No locking hierarchy or file-open functionality
  - No way to pre-declare an intended file *access-mode* before reads/writes, or a *deny-mode* for others accessing the file
  - No way to obtain a file-lock prior to requesting a byte-range lock.

- Advisory Locking
  - System does not validate **read, write, create**, **rm**, **rmdir**, **mv**, .. against locks
  - Enforcement of locks relies on compliance by well-behaved applications.

Issues Impeding CIFS, NFS/NLM Interoperability

A. CIFS *Hierarchical* Locking vs. NFS/NLM *Non-hierarchical* Locking

B. CIFS *Mandatory* Locking vs. NFS/NLM *Advisory* Locking

C. Server OS (e.g. UNIX) may lack means to validate
(local or NFS) **read**, **write**, **create**, **rm**, **rmdir**, **mv**, ..
vs. CIFS locks

Problems 1-3 with Uncoordinated Concurrent Reads and Writes arise in the mixed CIFS, NFS/NLM environment if these issues are not dealt with, i.e.:

1 Readers receive stale data

2 Writers overwrite each others' updates

3 Applications have in-use files deleted or renamed "out from under" them

Features of SecureShare = Multiprotocol Lock Manager

- **Multiprotocol Data Integrity**

  Reconciles the different and incompatible locking and file-open semantics utilized by CIFS and NFS/NLM clients.

- **Multiprotocol Oplock Management**

  Supports standard CIFS oplocks, while at the same time making oplocked data available to NFS-based clients through multiprotocol oplock break.

- **Multiprotocol Change-Notify**

  Supports standard CIFS *change-notify*, while extending it to cover changes due to NFS in addition to covering changes due to CIFS

The Uniform Lock-Mode Model

- Uniform *lock-mode* encompasses both
  file-locks and *byte-range locks*
- *Lock-mode* expresses exclusivity of access:
  lock-mode = *access-mode* “+” *deny-mode*
- **Open** --> **File-lock**:
  lock-mode (**file-lock**) = *access-mode & deny-mode* (**Open**)
- Byte-range locks:
  read-lock = non-exclusive = Read/Deny-Write
  or
  write-lock = exclusive = Read-Write/Deny-All

Coping with NFS/NLM's Lack of File-Open

- **file-lock**'s *deny-mode* vs. **NLM byte-range lock**'s *access-mode* (approximates NFS/NLM's "Open" *access-mode*)

- Treat **NLM byte-range lock**'s *deny-mode* as Deny-None (only applies to byte-range, not whole file)

- Example: New **Open**/Deny-Read or Deny-Write or Deny-All fails if pre-existed exclusive **NLM byte-range lock**

- Example: New exclusive **NLM byte-range lock** request fails if pre-existed **Open**/Deny-Read or Deny-Write or Deny-All

The CIFS Oplock Model

- Oplocks Assure Global Cache Coherency for Read-Write-Shared Files with Minimized Network Traffic / Maximized Client Caching

- Server “Opportunistically” Grants First Client’s **Open** (though non-exclusive) a temporary (“*breakable*”) **exclusive** file-lock

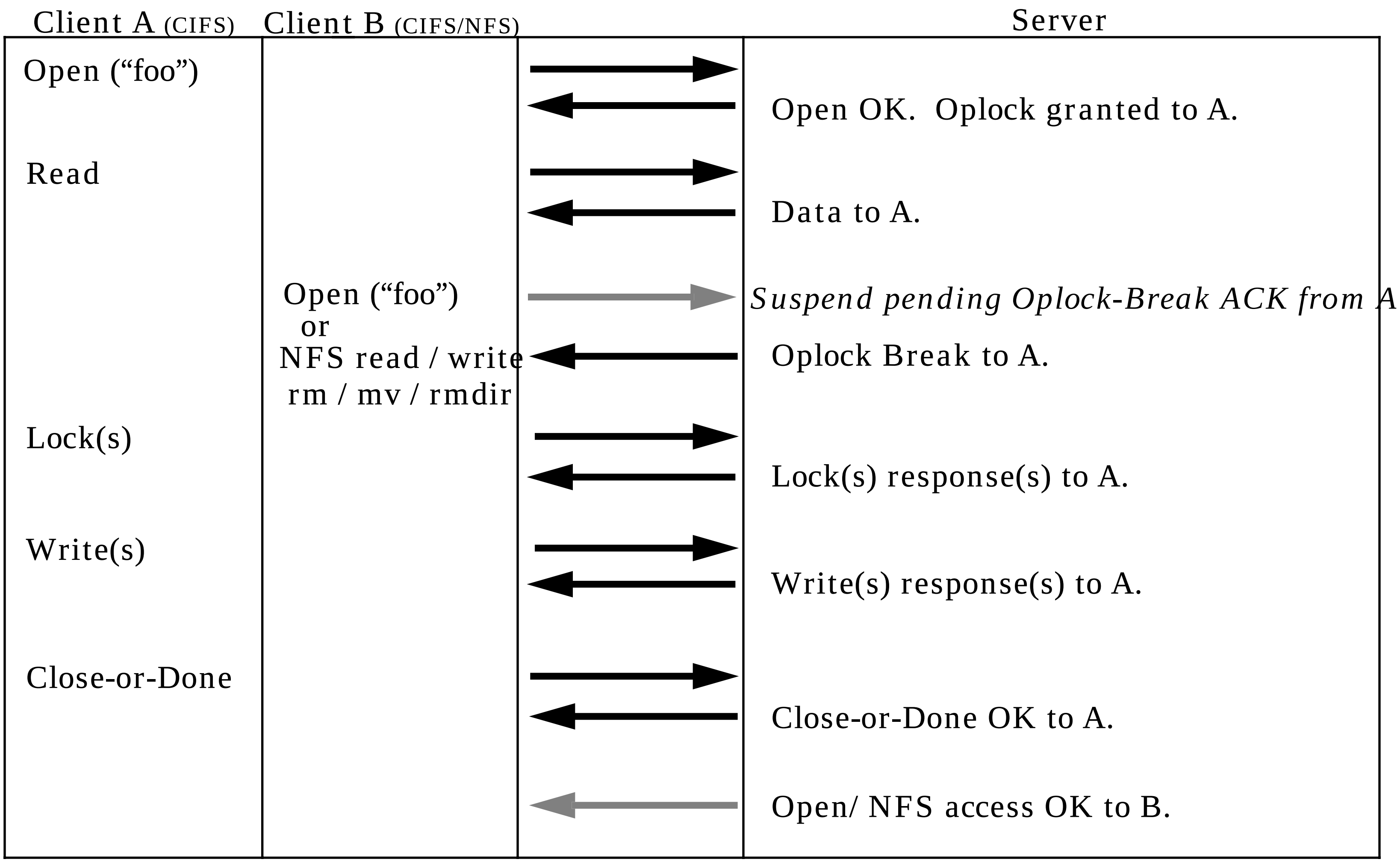
- Client Caches **Writes**, **Locks**, **Readaheads**;
  Batch Oplock (kept “forever”): Client Caches Application **Opens**, **Closes**.

- Second Client’s **Open** is Suspended while
  Server Sends Oplock Holder an ***Oplock-Break-Message***

- Client Holding Oplock Has a Choice:
  (1) **Close** the File (e.g. “stale” *batch* oplock: application has exited); or
  (2) Flush Cached **Writes** & **Locks**, send ***Oplock-Break-Ack*** message

- Server Now Allows the Second Client’s **Open** to Proceed

How Oplocks Work



Client A (CIFS)

Client B (CIFS/NFS)

Server

Open ("foo")

Open OK. Oplock granted to A.

Read

Data to A.

Open ("foo")
or
NFS read / write
rm / mv / rmdir

Suspend pending Oplock-Break ACK from A

Oplock Break to A.

Lock(s)

Lock(s) response(s) to A.

Write(s)

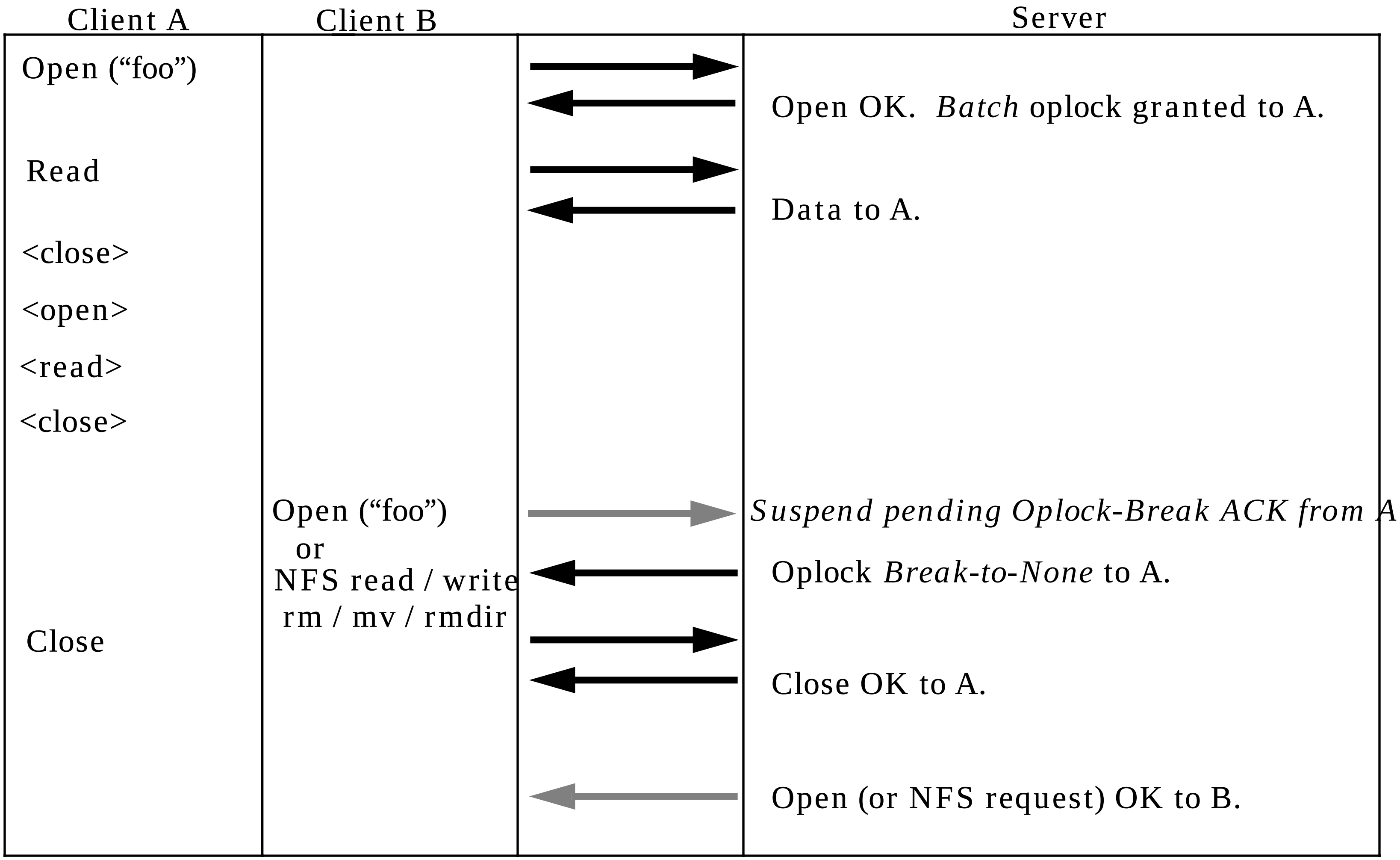
Write(s) response(s) to A.

Close-or-Done

Close-or-Done OK to A.

Open/ NFS access OK to B.

How Oplocks Work (Batch Oplock)



Client A	Client B		Server
Open ("foo")		→	
		←	Open OK. *Batch* oplock granted to A.
Read		→	
		←	Data to A.
<close>			
<open>			
<read>			
<close>			
	Open ("foo") or NFS read / write rm / mv / rmdir	→	*Suspend pending Oplock-Break ACK from A*
		←	Oplock *Break-to-None* to A.
Close		→	
		←	Close OK to A.
		←	Open (or NFS request) OK to B.

Why NFS and NLM Must Break Oplocks

- Choices when NFS or NLM encounters an ***oplocked*** file:

  #1 **Enforce** potentially breakable oplock ----->
  file is unnecessarily ***unavailable*** to NFS/NLM applications

  #2 **Ignore** the oplock ----->
  imperils file's ***data integrity***

- Choice #1: Unreasonable ***unavailability*** to NFS/NLM in cases

  (a) ***Stale*** *batch* oplock: Application closed file hours ago!!

  (b) Unnecessarily exclusive file-lock: Still-current open was **non-exclusive**

- Choice #2: NFS operation could lead to ***data corruption***

Example: NFS ***rm*** Encounters an Oplocked File...

- NFS **rm** *Suspends* during Oplock Break Send-Response, then *Restarts*

- Case of Stale *Batch* Oplock:
  Oplocker Responds: File **Close**
  (*Restarted*) NFS **rm** Succeeds

- Case of Application Still Using File:
  Oplocker Responds: Writes, Locks, Oplock-Break-Ack
  (*Restarted*) NFS **rm** Fails

Conclusions

Need Integrated NLM / CIFS Lock Manager to

- Maintain CIFS Data Coherency when CIFS and NFS Share Read-Write Access to the Same Files
- Prevent Files from Being Removed/Renamed "out from under" a PC Application
- Allow CIFS Clients to Receive **Oplock Break** Notifications when NFS Attempts to Access an Oplocked File
- Send **Change-Notify** Messages to an NT GUI Window when NFS Makes Changes in a Directory