High Level Design for CIFS Parallel I/O Filter

Matt Wu

2005/09/07

1 Introduction

To simplify lustre windows porting, CIFS parallel I/O filter is proposed. First we need export lustre mount points with Samba. Then windows system could access the Samba server easily with all lustre components bypassed. Via this way all the data have to be transferred between the windows client and the Samba server. The connection between these two servers will be a bottleneck. So we need a new windows file system filter driver to implement parallel I/O, i.e. redirect the data I/O requests to OST servers. The filter driver needs to query the stripe distribution layout and then split the whole I/O into pieces and issue the small requests to different servers where the OSTs locate. Then the second step, let Samba directly exports the OST device and windows client could directly read or write the exported OST partition. The third step is to create a network file system driver of lustre, Samba and also the filter driver are to be deserted then.

This document only covers the high level design of the filter driver for the first step.

2 Requirements

- Capture the I/O to MDS and redirect the requests to the corresponding OST servers

3 Network File System Architecture

Windows manages all the network resources in a unique namespace with UNC (Universal naming convention) standard. The UNC names begin with the characters "\\", which is indicating the resource exists on the network.

Windows network file system driver contains several components:

- Network provider

- MPR (Multiple Provider Router)
- Lan Manager in user mode
- Mup (Multiple UNC Provider)
- File system redirector (it includes the network redirector driver and the LanMan file system wrapper)

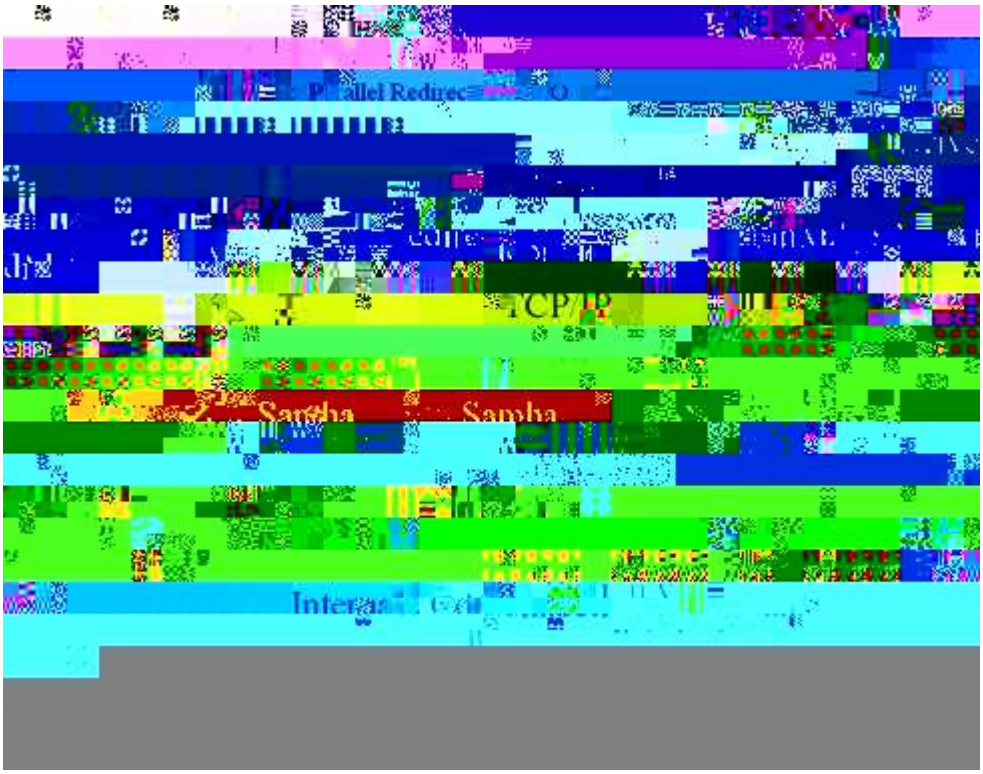
Different network file system has different network provider and the redirector driver. The provider handles the connections to the remote network server. When an application calls WNet API routines to operate a shared resource, it will be passed directly to the MPR. Then the MPR dll library will take the call and determines which WNet provider recognizes the resource being accessed. Then the requests will be transferred to kernel space in the end. It's MUP that will determines which local redirector recognizes the remote resource which could be a file or a device with a UNC name. Thus the callback routines of the redirector driver will be called to process the request.

4 Parallel I/O Filter Architecture

For every server node, we need start the lustre client: llite to mount it as /mnt/lustre. The Samba should share the lustre mount point in the following rules:

Samba Server	Mount Point (Ex)	Sharing Name (as UUID)	Windows Referrence Path
MDS server	/mnt/lustre	\\mds\lustre	\Device\LanmanRedirector\mds\lustre
OST1 server	/mnt/lustre	\\ost1\lustre	\Device\LanmanRedirector\ost1\lustre
OST2 server	/mnt/lustre	\\ost2\lustre	\Device\LanmanRedirector\ost2\lustre
...	...	...	...

Then for client side, windows system could connect to every Samba server via CIFS protocol. The content of \\mds\lustre, \\ost1\lustre and \\ost2\lustre ... should be the same. But windows CIFS client could not understand these circumstances. For a system with 1 MDS + 2 OSTs as an example, if a file stripes over OST1 and OST2, when windows client tries to read it's content, the MDS would read the data from OST1 and OST2 via portals protocol, then send it back to windows side. All the I/O traffic is to be transported between windows client and the MDS server.



With the parallel I/O filter driver, it will capture the data I/O requests to the MDS Samba server and redirect the requests to the corresponding OST servers according to the file stripe distribution, see the dark red lines in the picture above. All the meta-data requests will be done on MDS and the real data I/O requests are to be performed on the OST servers. The network traffic will scatter between 3 paris instead of 1 between MDS server and windows client.

5 Names Management

5.1 Functional Specification

Our filter driver will filter the LanMan Redirector (NwRdr), thus monitors all the network operations, that might be a huge traffic. We must figure out the I/Oes we concerned. The way is to compare the UNC name to check if it has the prefix of the MDS share path. Ex: "\Device\LanmanRedirector\mds".

The full path name collection is normally to be done when the file is first opened, i.e. in the IRP_MJ_CREATE request handler routine. We need query the full path of the specified file and store these information in a generic table or a hash list for later referrence, because in later I/O requests there's no way to query the full path name for some cases. With the generic table we can get

the path name of the file to construct the names on OST servers.

The Fcb pointer (FileObject->FsContext) could be treated as a unique key magic for every file. It's stored in pair with the full path name. All the names with the prefix "\Device\LanmanRedirector\mds\" should be recorded by the filter driver. Then for later I/O requests, the names on OST should be constructed from it, just like "\Device\LanmanRedirector\ost1\a.dat", "\Device\LanmanRedirector\ost2\a.dat". With the full path name, we can redirect the I/O requests to the corresponding OST servers.

There also another issue on naming management: rename operations. We need trace the renaming operations to update the file name modifications, otherwise the I/O to the newly renamed files will be missed.

5.2 Use Case

N/A

5.3 Logic Specification

- Structure for Name Entry

```
typedef struct _NAME_ENTRY {
    USHORT         Magic;     /* Magic */
    USHORT         Flags;     /* Flags */
    ULONG          RefCount;  /* Refer count */
    PVOID          FsContext; /* Key (Fcb) */
    UNICODE_STRING Name;      /* Full path name in unicode */
} NAME_ENTRY, *PNAME_ENTRY;
```

The structure of NameEntry and name string buffer should be allocated from the NonPagedPool to ensure the safe access under IRQL > PASSIVE_LEVEL. Generic table will be used to manage the name entries. System already provides the runtime routines to operate on generic tables.

- Creation: Query Full Name in IRP_MJ_CREATE

We can deduce the file names from the user specified parameters, but only for limited cases. So we'd better issue a name query request after the create/open irp is completed successfully, then complete the irp maually to return to the upper filters or the I/O manager. The sfilter example in ifskit realizes it via a kernel support routine: ObQueryNameString. We can reuse it for our purpose.

- Update: Trace Renaming in IRP_MJ_SET_INFORMATION / FileRenameInformation

This part is not realized in sfitler source. We need implement it on our own. There are two core structures related for the rename operation: IO_STACK_LOCATION.Parameters.SetFile and FILE_RENAME_INFORMATION structure stored in Irp->AssociatedIrp.SystemBuffer.The renaming process could be divided to 3 cases deponding on the complexity:

1. Simple Rename: SetFile.FileObject is NULL.
2. Fully Qualified Rename: SetFile.FileObject is non-NULL and FILE_RENAME_INFORMATION.RootD is NULL.
3. Relative Rename: SetFile.FileObject and FILE_RENAME_INFORMATION.RootDir are both non-NULL.

For case 1 and 2, we can get full pathes for the old and new file names in no trouble. But for case 3, we need some extra processing to retrieve full pathes and do the substitution of NameEntry.Name inside the generic table.

- Destruction: Release the buffers in IRP_MJ_CLOSE

When the file is no longer used, system will issue an IRP_MJ_CLOSE to the opened file. If it's the last reference, all the information in memory of the file will be destroyed. So that's the just time we need remove the NameEntry from the generic table and release the memory of the buffers.

5.4 State Machine

- Access to the generic table should be under global lock protection.
- NameEntry's lifecycle is controlled by the reference count (NameEntry.RefCount). The referrence count is to be increased by 1 in IRP_MJ_CREATE and decreased in IRP_MJ_CLOSE. When the referrence count becomes ZERO, we need release the NameEntry to system memory.

6 EA Operations

6.1 Functional Specification

The stripe distribution information is to be stored in "lov_dist" EA for every inode and the Samba servers information (ip address, netbios name, UUID) is to be stored in "ost_map" EA of root inode. The details are in the HLD / DLD documents of "LOV EA Support".

Windows system need implement the EA query routines to get the content of these EAs via Samba server. These jobs are already done during the ext3 protocol test. Also a patch for Samba was made then.

All the EA querying jobs are done via a kernel support routine: ZwQueryEaFile.

6.2 Use Case

N/A

6.3 Logic Specification

1. Open the file with proper user's context (see next seciton: Security Support) with ZwCreateFile
2. Initialize parameters FILE_FULL_EA_INFORMATION and FILE_GET_EA_INFORMATION
3. Call ZwQueryEaFile to query the whole EA list or the value of a specified EA

6.4 State Management

N/A

7 Security Support

7.1 Functional Specification

As we know, the CIFS sharing protocol needs a user name / password certification to access a specified network resource. Only after loging on correctly could we access the shared resource.

We can do it manually and make system cache the certification information, then we can bypass the manually logon process when trying to access the shared resource. We could also write a user program using WNet routines (WNetAddConnection) to logon the Samba server automatically to privilege the user to the access of the remote servers.

But for any case, for our filter driver, only in that specifiy process context, we are privileged. If we want to operate on the shared resource on a different context or thread, the security manager might deny our requests.

Fortunately windows system provides us a powerful technique to accomplish the impersonation. With this technique, we can store the credentials when we are in the user's context (normally in IRP_MJ_CREATE) and restore it in other circumstances as like the operation is just performed by the user itself.

7.2 Use Case

N/A

7.3 Logic Specification

1. the first request to open/create file (IRP_MJ_CREATE) is in the user's context. It carries the user's security token. At this time we could create the SECURITY_CLIENT_CONTEXT with routine SeCreateClientSecurity to store the user's credentials. We could also do this process in an ioctl handler routine and store it for global usage.

2. when we want to access the restricted resources in other context, we need impersonate the user's context. The routine SeImpersonateClientEx does this for us.

3. after the restricted operations are done, then restore the context to the original one by calling PsRevertToSelf.

7.4 State Management

We can use the SECURITY_CLIENT_CONTEXT as a global information. Then no need to care the state. If we wants to use one security context per file (NameEntry), we can create and destroy the seucrity context according to the lifecycle of NameEntry.

8 Parallel I/O Dispatch

8.1 Functional Specification

The core I/O functions are processed by IRP_MJ_READ / IRP_MJ_WRITE handler routines. For our redirecting purpose, it's ok only to redirect noncached I/O (including paging I/O). The cached I/O are to be ignored.

The typical procedure of windows I/O is like the followings:

- Cached Reading Process:

1. User issues a reading request.

2. IRP_MJ_READ (cached I/O) is to be called ultimately to the file system driver.

3. The file system driver should initialize the cache support for the file if it's not initialized yet. Then the cache manager will enable the read ahead behavior for the file object.

4. Fsd will call CcCopyRead to prepare the pages for the request and return to the user. Attention here: at the monment there's no any data manipulation yet.

5. Then the user will try to access the pages, which will cause a page fault.

6. Then the page fault will get to the fsd. IRP_MJ_READ is called again, with Paging I/O (NonCached I/O) flags set.

7. The fsd should perform the disk I/O to read data from disk to system memory (cache).

8. After that's done, system could restore from the page fault handler to the user. Then the user's reading request is satisfied.

- Cached Writing Process:

1. User issues a write request.
2. Fsd routine IRP_MJ_WRITE (cached I/O is to be called.
3. Initialize the cache support for the file if it's not initialized. Then the cache manager will enable the read ahead / laze write behavior for the file object.
4. Fsd will call CcCopyWrite to prepare the pages and write data into the cache pages, then return to user.
5. User is notified that it succeeds to write data. (But now the data is still in the cache.)
6. The cache manager will issue a request of IRP_MJ_WRITE (paging I/O, non-cached) to write all the cache into the underlying disk devices.
7. Then file system driver will perform the disk I/O to write data to the disk. All the writing process is finished.

These two handler callbacks could be called at both at irql PASSIVE_LEVEL and APC_LEVEL (paging I/O). The ZwReadFile/ZwWriteFile routines are forbidden to be called for any IRQL other than PASSIVE_LEVEL. And there are restrictions on these two routines even we use WorkItems. We'd better construct our own Irp with high flixibility and issue them to the OST servers.

8.2 Use Case

N/A

8.3 Logic Specification

1. Check whether the file I/O is to the MDS server ? (Is FileObject->FsContext in the global generic table ?)
2. If yes. Then construct the object path names of the OSTs. EA querying might be needed to get the ost_map EA. Security context should be impersnated before the EA querying call.
3. Then query the file's stripe distribution information from MDS server.
4. Parse the stripe distribution information and split the memory block into pieces.
5. Impersonate security context if needed, open the file objects on the OST servers.
6. Construct irps for the opened file objects (calling IoAllocateIrp ...)

7. Issue the sub requests to the OST servers. We could get the device object from the file's FileObject.
8. When the last sub irp is completed and none of them fails, then complete the original irp with success.
9. If one of them fails, then do the failover procedure: retrieve the ost_map and lov_dist, jump to step 4.

8.4 State Management

N/A

9 Focus of Inspection

1. The design is reasonable ? Could be better ?
2. Could there be possible DLM deadlocks ?
3. Possiblity of stale cache in client side.

10 References

1. Help documents of Ifskit 2003
2. OSR documents in Ifskit 2003
3. Windows NT File System Internals by Rajeev Nagar
4. Inside windows 2000 3th by David Solomon and Mark Russinovich
5. Documents on http://www.osronline.com
6. Lustre book