High Level Design for Lustre Parallel CIFS Driver

Matt Wu

2006/01/11

1 Introduction

The new Lustre Parallel CIFS Driver will use osr filter kit to implement the parallel I/O dispatching instead of the original desgin with ifskit sfilter for the demo project.

The principle is the same to sfilter design. So here it's enough just giving a brief description on the basis of the filter driver architecture. The lustre linux servers (mds and osts) are to export the lustre client to windows via Samba. The windows system with the parallel driver running will redirect all the I/Oes which are to be sent to mds server to the ost servers directly to improve the degree of paralelism and the network I/O throughput.

2 Requirements

- Capture the I/O request to MDS and redirect to the OST servers where the file data lies in
- Productize the Lustre Parallel CIFS Driver with osr fddk

3 Osr FDDK Architecture

FDDK contains 3 components (the functionality driver is implemented by the user):

- Osr Recognizer
- Osr Filter
- Functionality Driver (pCIFS)

The recognizer driver mainly focuses on file system recognition progress, i.e. loading or unloading of file system drivers, new volumes insertion or media /

disk removal. It filters the file system driver's management device object to monitor all the behaviors on volumes.

In the contrary, the osr filter driver only focuses on the file operations on a specified volume. It provides several callback routines to recognizer. Then the recognizer will trigger the callback routine to filter news volumes being mounted if the volume is under our interest.

The functionality driver is the just pCIFS driver of our project. We need register callback routines just to hook several requests during initialization, such as IRP_MJ_CREATE, IRP_MJ_READ/WRITE. The osr filter will execute our callbacks to process the original request when the conditioins meet the needs.

There are about 7 types of callbacks according to the structure OSR_FILTER_CALLBACK. We'll concern the former 4 types. The latter 3 could be ignored in our pCIFS funcitonality driver.

The OSR_FILTER_CALLBACK structure in "inc\osr-filter.h":

```
//
// DLL registration data structure.
//
typedef struct {
    OSR_FILTER_DISPATCH_PROC        Dispatch[IRP_MJ_MAXIMUM_FUNCTION+1];
    OSR_FILTER_COMPLETION_PROC      Complete[IRP_MJ_MAXIMUM_FUNCTION+1];
    OSR_FILTER_MOUNT_PROC           Mount;
    OSR_FILTER_DISMOUNT_PROC        Dismount;
#ifdef FDDK_FILTER_FAST_IO
    OSR_FILTER_FAST_IO_PROC         FastIo;
#endif // FDDK_FILTER_FAST_IO
    OSR_FS_FILTER_NOTIFICATION      FsFilterNotificationCallback;
#ifdef USING_PNP_NOTIFICATION
    OSR_FS_PNP_NOTIFICATION         FsPnPNotificationCallback;
#endif // USING_PNP_NOTIFICATION
} OSR_FILTER_CALLBACK, *POSR_FILTER_CALLBACK;
```

1. Dispatch: Normally IRP filter routine, to be called before the Irp is transferred to the underlying filesystem driver.

2. Complete: Normally IRP filter routine, called when the original Irp is completed (processed).

3. Mount: To be called when new volumes are to be mounted.

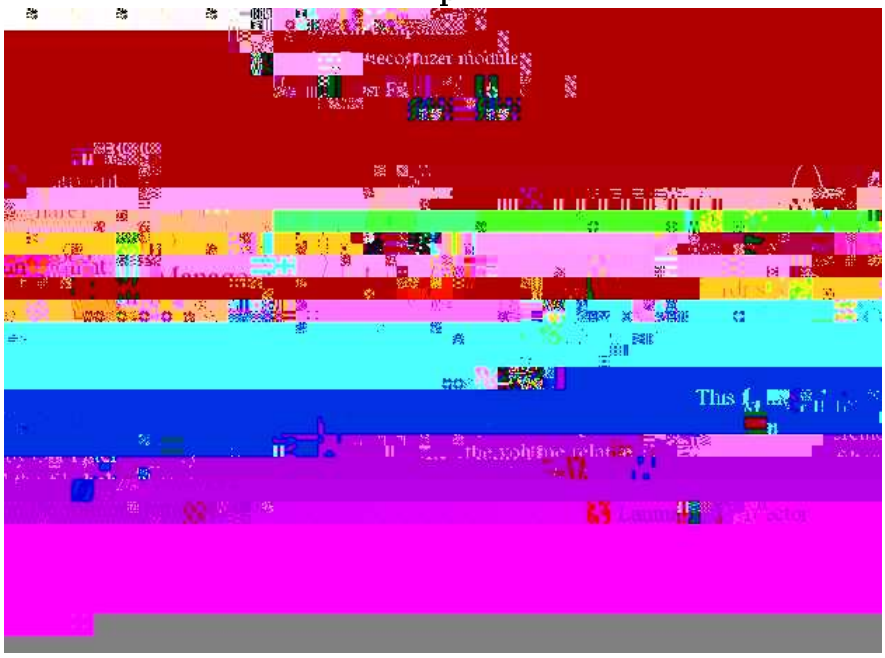
4. DisMount: To be called when dismounting.

5. FastIo: To hook the FastIo routines. The FastIo routines are introduced for performance consideration. It bypasses context switching, irp construction.

6. FilterNotification: Notificaiton callbacks for cache modification or FastIo operations.

7. PnpNotification: To be notificed of the Pnp events, such as Volume change, mounting, dismounting.

Picutre of the osr fddk components:



4 pCIFS Callback Handlers

We need provide 3 types of callback routines when registering OSR_FILTER_CALLBACK:

1. Mount and DisMount
2. Create: Complete
3. Read/Write:Dispatch

4.1 Mount and DisMount

Here we need initialize the globals for management when mounting and clean up the structures when dismounting. The data structures manipulated are the management context information of the cluster, please refer section 5 for details.

4.2 Create Completion Routine

Here we only care the result of the IRP_MJ_CREATE process. If the file is opened or created successfully and it's belongs to MDS share, we need create our own context data for later usage. The context information mainly contains

full path name, security context, stripe distribution layout information, etc. In section 6, we'll describe them in detail.

4.3 I/O Dispatch Routine

This is the the core routine of I/O redirecting. We need completely take over the handling of IRP_MJ_READ and IRP_MJ_WRITE. Thus we need provide the dispatch callback routines for Reading & Writing and return STATUS_PENDING to the osr filter driver. That status code tolds the osr filter that the default handler routines are to be bypassed.

5 Context per Cluster (servers group)

5.1 Functional Specification

These context information are global for a lustre cluster. Generally it contains two types of data:

- Lustre servers configurations
- Runtime management globals

Previously in sfilter demo project, we are using global structures to maintain these configurations. In the new design we'd better dynamically allocate it as private volume user context of osr fddk when mounting new Lustre MDS. Thus the filter driver could support more than 1 cluster groups at the same time without any confliction. The context is to be created in Mount callback and destroyed in DisMount callback.

1), Lustre Servers Configurations

The configuration is the just Lustre servers layout and the shared names. Windows could access linux samba export via different names:

1. Samba shared names (host name, samba netbios name)
2. Mapped driver letters
3. Server ip addresses

Normally different names of a signle server will be treated as different servers, i.e. there will be different volume objects in the eye of the file system driver or filter driver. So some I/Oes may bypass our monitoring and step into another path.

For the method 2, we can trace the original shared path form the driver letter. Osr fddk already does this resolving for us (in GetFileLongName in filter/filterdispatch.cpp). The remained job and other naming menthods should be done by oursolves. That needs us to maintain a table to store all the names

of a server. In case of MDS server, all requests via possible MDS names and the mds ip addresses will be treated as the same and redirected to the corresponding OST servers. These settings could be obtained via EA of ROOT directory or special configuration locating in the root, such as ".SRV_MAP", we are planing to use the latter method here.

2), Runtime Management Globals

This part is to maintain the online various structures, such as memory lookaside lists, statistic for performance/memory usage, core structures back trace list.

5.2 Use Case

We create and initialize the cluster context during mounting. But there's one difference on which the LanmanRedirector differs to local file systems: all the remote servers are treated as a signal file system with UNC names. That means the whole UNC volume is only mounted once and there's only one dismount in corresponding. For a local file system, the driver will get one mount request for every volume during it's first access. At the time of LanmanRedirector mounting, no any real remote servers are connected yet. We have to trace the request of IRP_MJ_CREATE to catch the first access of the specified remote lustre share.

Example: In the PostDoCreate callback routine, i.e. the completion routine of the IRP_MJ_CREATE request, if the full path name if "\\Device\\LanmanRedirector\MDS\lustre\", we treat it the mouting time of the lustre cluster. Then read the content of ".SRV_MAP" and construct all the configurations.

After the cluster context is created, every file on the cluster will keep a trace pointer to the cluster context and increase the cluster context's refercount. Then in later parallel I/O routines, we could get the cluster context with the file's stream context and construct the file names on different ost servers and issue the sub requests to different ost servers.

5.3 Logic Specification

1), Linux:

Previous we designed to store the configuration information in an EA of the root inode, but it won't convenient to process server configuration change such as new ost server being added into the cluster. To overcome the problem, we are planing to use a file named ".SRV_MAP" in the lustre root instead.

The file is to be created when llite is mounting, with the content of the servers configuration, and to be updated when the configuraiton is changed. Thus a directory notification on the root is to be sent to the windows client to notify of the change. Then windows client should discard the current configuration settings and reconstruct with the new content of ".SRV_MAP".

*** a), Content of the ".SRV_MAP" under lustre root directory**

```
/* samba server map information (mds and ost) */
#define SF_MAX_ALIAS_NUM   (8)
#define SF_MAX_ALIAS_LEN   (32)
#define OST_INFO_MAGIC 'OI'
/* information per ost*/
struct ost_info {
    /* magic and flags */
    __u16  magic;
    __u16  flags;
    /* ost index number */
    __u32  ost_index;
    /* ost uuid string, trailing with NUL */
    const  char uuid[40];
    /* ost alias name, max 32 chars in length, 8 in number */
    const char alias[SF_MAX_ALIAS_NUM][SF_MAX_ALIAS_LEN];
    /* ip address of the ost server */
    __u32 ipaddress;
};
#define SRV_MAP_MAGIC  'SM'
/* the content format of file .SRV_MAP */
struct srv_map {
    /* magic and flags */
    __u16  magic;
    __u16  flags;
    /* number of ost devices */
    __u32  osts_count;
    /* mds alias name, max 32 chars in length, 8 in number */
    const  char alias[SF_MAX_ALIAS_NUM][SF_MAX_ALIAS_LEN];
    /* array of ost information */
    struct ost_info osts_info[0];
};
```

*** b), creation and updating**

In linux side we'll use the configuration tool to create and update the configuration file on root directory.

2), Windows

The MDS configuration should be told as a parameter to the Windows filter driver. Then all the other necessary configuration information could be gained via the EA of SRV_MAP.

In windows we need maintain a list of MDS servers (share point) in both the registry and the driver. The list in registry is only to keep the setting in

system, which won't be lost after rebooting. The list in driver is created during runtime containing all the active clusters. The window configuration utility is in charge the list. When new cluster is inserted or one cluster is to be removed, the utility will talk with the filter driver of that change.

The item of the MDS list in driver should contain the full path name, ex: \Device\LanmanRedirector\mds\lustre. The server management routine should keep the SRV_MAP information for every MDS entry. The MDS list should be a driver-global structure, but the SRV_MAP info should be a volume-global structure. The former is always alive while the driver is loaded; the latter is valid when the volume is being mounted. Our filter driver will be kept resident all the time, so no lifecycle management for the MDS list. But for the server configuration (SRV_MAP) could be created when mounting and cleaned up when dismounting.

For the runtime management contextes, most of them could be treated as globals, such as memory allocation lookaside lists. The other volume-specified could be treated the same way to SRV_MAP.

5.4 State Machine

*** a), Creation of the cluster context**
The context is to be created when

*** b), Destruction of the cluster context**
We'll draw an extra reference on the cluster context when creating the stream context for every file. The file's stream context is to be released automatically by system. During destruction of the stream context, it will drop the reference of the cluster context. If there's no reference on the cluster context, we need release the cluster context.

*** c), Handling server configuration changes**
When constructing the cluster context in PostDoCreate, we'll start a system thread to monitor the notify changes of the root directory of the MDS share. Windows kernel provies a routine ZwNotifyChangeDirectory to query the change notify information.

After querying, we need parse the information block to trace whether ".SRV_MAP" is modified or not. The configuration file could be modified only when the configuration is changed.

Once we get the configuraiton file is changed we'll purge all the configuration context in memory and update them with the new configuration information.

6 Context per File Stream

6.1 Functional Specification

Osr maintains the unique context for any opened file, i.e. stream. The context is to be released when the file is closed internally. In the pCIFS driver, we need not care the lifecycle of the per stream context, osr fddk does it for us. The routines to access the per stream context is OsrGetPerStreamContext.

Several context information should be maintained in the per stream context for our purpose:

1. file full path name
2. security impersonation context
3. the LOV stripe distribution layout
4. other runtime management information

* a), Context/Names Management

The file full path name includes the "\\Device\\LanmanRedirector" prefix. With that long name we could decide whether the request is sent to MDS or not, then construct the corresponding OST target full names to be redirected to.

The osr fddk implements the name cache internally. It handles rename operation as well. With the compiler switch: FDDK_FILENAME_CACHING turned on, the file full path information will be stored in the OSR_PER_FILEOBJECT_CONTEXT, which could be obtained via OsrGetPerFileObjectContext.

As we mentioned above, the deivce name in file full path might be not identical for a server may have different names. We need do the extra process in the post-create callback routine to make the share names identical. If the request is sent to MDS, we need create our own specified context and keep it in our own list for later usage when processing I/Oes. We also need create the security impersonation context and query the necessary EA informaiton for the requests of our concern, these things are to be detailized in their own subsections.

* b), Security Impersonation Context

After IRP_MJ_CREATE is successfully completed by the underlying file system driver, we can store the operator's security context and impersonate the specified user in other context such as when trying to open the handles of the OST target names in IRP_MJ_READ or IRP_MJ_WRITE.

* c), LOV Stripe Distribution Layout

In previous design we implemented a new EA for every lustre inode: "lov_dist". We also could let llite export the original EA value "trusted.lov" to users, and it surely does in b1_4.

The routines of querying or setting an EA are already implemented. The EA operation is to be done in IRP_MJ_CREATE, though the value is be referred in IRP_MJ_READ and IRP_MJ_WRITE requests handler routines, because it will cause a deadlock hang if we issue EA requests in the reading or writing handler routines which could be executed at APC_LEVEL. The stripe information will be stored in the per stream context and could be easily gotten at any time.

6.2 Use Case

The stream context of every file is constructed in the PostDoCreate routine and to be referred in the parallel I/O routines. The I/O routines will divide the user's request into sub requests due to the file stripe distribution.

The fddk is to destroy the stream context with the destruction routine we registered when creating the stream context.

Osr fddk provies routines to register and query the context pointer for every stream.

6.3 Logic Specification

The context creation is done in the osr completion callback of IRP_MJ_CREATE. The user context structure could be defined as the followings:

```
/* file management context for every opened stream */
/* magic & flags definitions */
#define PCIFS_STREAM_CONTEXT_MAGIC 'PSCM'
typedef struct _STREAM_CONTEXT {
    ULONG                       Magic;      /* Magic */
    ULONG                       Flags;      /* Flags */
    ULONG                       RefCount;   /* Refer count */
    PFSRTL_ADVANCED_FCB_HEADER  Fcb;        /* FsContext */
    FOSR_PER_STREAM_CONTEXT     Context;    /* osr per stream context */
    PSECURITY_CLIENT_CONTEXT    SCC;        /* Security Context */
    UNICODE_STRING              Name;       /* Full path name */
    struct lov_mds_md *         lmm;        /* lov_mds_md ... */
    LIST_ENTRY                  Link;       /* linked into global */
    ......
} STREAM_CONTEXT, *PSTREAM_CONTEXT;
```

* a), Security Impersonation Context

This part is the same to the previous pCIFS design.

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.

2. when we want to access the restricted resources in other thread 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.

* b), LOV Stripe Distribution Layout

1. Open the file with proper user's context (see next seciton: Security Support) with ZwCreateFile

2. Construct our own IRP_MJ_QUERY_EA Irp to query the content of "trusted.lov" from MDS share

* c), Context/Names Management

1. Query the fullpath name and check if it belongs to the MDS share

2. Create the user context if the file is not filterred yet

3. Store the full path name to our user context newly created

4. Store the current user's "Security Context" to the user context

5. Query the content of "trusted.lov" from MDS and store it for later usage

6.4 State Machine

1. these structures are created when first filtering the file in IRP_MJ_CREATE completion callback

2. to be destroyed in osr filter callback of freeing the user context

3. the access of these structures should grab the reference count and release it after referring

7 I/O Dispatch

7.1 Functional Specification

The new design is most simliar to the demo project: we take over the complete control of I/O requests and split the original Irp according to the stripe distribution layout and send the sub requests separately to the corresponding OST servers.

7.2 Use Case

N/A

7.3 Logic Specification

1. Call OsrGetPerStreamContext to query the private user context.
2. The user context (STREAM_CONTEXT) should contain the full path name,security context... if it belongs to the MDS share.
3. Then queue a workitem to lower the IRQL to PASSIVE_LEVEL under which level ZwXXX routines could be executed. And returns STATUS_PENDING to osr filter. Then osr filter will stop any extra processing and return to system.
4. The workitem callback routine will call RealReidrectIO to redirect the request to OST shares.
5. ReadRedirectIO will parse the lov distribution layout and split the big request into several smaller ones and issue them to the corresponding OST servers.
6. When all the split small Irps are completed, the completion routine will collect the results and complete the original big I/O request.

7.4 State Machine

N/A

8 Cache Issues

With parallel I/O filter dirver, there is big possibility of causing cache coherent problem, since system maintains different cache copies for the OST share instances and the MDS share instance. For lustre they are the same file at all, but for windows aspect, they are completely different.

When completely redirecting Reading and Writing requests, there's no I/O between windows client and MDS share at all. Then on MDS there's no stale cache problems, but on OST shares, there's still the possibility if the user trys to directly access the OST share point. The stale cache problem could occur upon with the special stripe patterns. We may need to purge the stale cache range if necessary. The purging could be done with windows kernel support routines: CcPurgeCacheSection and CcFlushCache. But the cache purging operation is a time-costing job and should be done with care otherwises it could easily make deadlocks.

In case of only redirecting Paging I/O and NonCached I/O, the cache purging is a must to keep coherent. Since the cached I/O are issued to MDS share

instance, but the real data is written to OST share instances. The cache of OST share instances won't have chance to update their cache.

9 Performance Improvement

The results of the demo pCIFS project shows that the reading performance is not good, the writing performance is reasonable. More analysis on lustre reading is to be made to find out the bottleneck.

A test was ever made on a computer with two network cards. Both of the network could get to the peak if using parallel filter to redirect reading to two identical in content but self-existent files on different servers.

Another way was ever tried: only redirect paging I/O or noncached I/O. But this way will cause stale cache problems. Besides the cache issues, there needs workaround to identify the cached I/Oes which won't trigger paging I/Oes as disk media file system driver does.

Possible optimizations on I/O process:

1. Using our own threads instead of system workitems to lower the IRQL. More efficient, less possibility of deadlocks.
2. Caching the opened handles of the corresponding OST file instances to remove un-necessary ZwCreateFile operations
3. Intergrating the small (ex: one stripe size long) requests into a single one. We need to cure the incontinuous offset / buffer problems since there are holes between the sub requests.

10 Utilities and Tools

10.1 Configuration Tools

- Linux: to configure Samba servers, Lustre, some internal settings such as create/modify EA
- Windows: to configure the registry settings for server names, user security management (Samba users authentication)

10.2 Test Tools

- Sanity test: Osr fddk batch test tool, filter test kit specified for filter drivers provided by ifskit.
- Performance test: the original performance test tool: parallel is to be enhanced.

11 Focus of Inspection

1. The design is reasonable ? Could be better ?
2. Performance issues and I/O throughput
3. Possiblity of stale cache in client side, especially when there are mulity windows clients attaching to the single cluster
4. Cluster configuration changes: such as ost migration, insertion or deletion

12 References

1. Ifskit 2003
2. OSR fddk
3. Lustre book, documents