Lustre Feature Test Plan for *Progressive File Layout*

Revision 0.3

April 6, 2017

Revision History

The following is a chronological history of changes made to this document.

Revision	Date	Reason for change	Author
0.1	March 2, 2017	Initial Version	James Nunez
0.2	April 3, 2017	Added list of tests in sanity-pfl, added performance tests and updated requirements to test suite/test number mapping	James Nunez
0.3	April 6, 2017	Corrected 'lfs getstripe' flag descriptions to reflect when a flag argument was optional or not. Removed –component-id flag from 'lfs find'	James Nunez

Table of Contents

Revision History 2

Introduction 5

Documentation 6

Feature Installation and Set-up 6

Regression Tests 6

Manual Testing 6

Interoperability 6

Performance Testing 7

User Interface 8

- lfs setstripe 9
- lfs find 9
- lfs getstripe 10
- lfs migrate 11
- User Interface Testing 12

API 16

- User Space API Use Cases 18

Feature Interaction 22

- HSM 22
- LFSCK 23
- Ost-pools 23
- Snapshot 23
- Subdirectory Mount 23
- Distributed Name Space (DNE) 23

What Not to Test 24

Results 24

Introduction

All regular files in a Lustre filesystem have a static layout that is determined at the time each file is first opened. Each file's layout specifies the number of stripes (OST objects) on which it stores data, between one and the number of OSTs in the filesystem. However, the file's layout cannot be modified after it is created, which forces the user to choose (or live with) the layout used at first open. If files are small, and/or many clients/threads are each writing to their own file (file-per-process, 1:1, N:N) then having a single stripe per file is optimal to reduce server-side contention and metadata overhead. If files are very large and/or accessed by many clients concurrently (shared-single-file, N:1, N:M), and/or have bandwidth requirements that exceed what is available from a single OST, it is desirable to have a large number of stripes for the file to maximize aggregate IO bandwidth and balance space usage across OSTs. While there are some mechanisms available to administrators, users, and applications to specify the layout per-filesystem, per-directory, or per-file for newly created files, this requires advance knowledge of how each file may be used (total size, number of clients writing to the file, required bandwidth, etc.) that may not necessarily be known to the application user. It is desirable that administrators, users, and applications be isolated from the need to specify the striping for each file in order to optimize IO performance for their various uses.

The Progressive File Layout Feature optimizes usage of both small and large files, avoiding the need for users to predetermine the required layout for each file. The goal is that a single file layout specified at file creation time can balance several conflicting needs:

- minimize the number of stripes for small files to avoid the overhead of creating and accessing multiple OST objects for such files
- maximize the number of stripes for large files to utilize the available bandwidth as well as distribute space usage to avoid consuming all of the space on a single OST
- flexibility to specify a layout that is suitable for a majority of uses, while allowing for variability and flexibility between users, applications, and sites

Further details of the goals, design and architecture of this project are listed in the Solution Architecture page at http://wiki.lustre.org/PFL2_Solution_Architecture.

The PFL feature is being tracked under JIRA ticket LU-8998.

Documentation

Man pages for 'lfs-setstripe', lfs-migrate, and 'lfs-getstripe' were modified to include the additional functionality added to these utilities for the PFL feature. Changes to the Lustre Manual are tracked under LUDOC-361 with a User's Guide attached to the ticket.

Feature Installation and Set-up

The Progressive File Layout feature is new to Lustre 2.10 and does not require any commands to enable the feature. The definition of the layout for component files can be set for the whole file system, for a directory or for a specific file with the use of 'lfs setstripe' or using the new API functions as explained below.

Regression Tests

The acceptance-small test suite will be run with a default layout specified for the file system to verify no functional regressions of existing testing. All tests are expected to pass with the exception of known failures. All testing shall be done with a combined MDS/MGS and a separate MDS and MGS.

Another goal of the PFL testing is to allow users to set a composite layout through the Lustre test suites so that all test suites can be run with a default file system composite layout.

Manual Testing

Manual testing may be required to test PFL's interaction with existing Lustre features as specified in the 'Feature Interaction' section. Any performance testing will be done manually.

Interoperability

Lustre clients prior to Lustre 2.10 will not able to access composite files. Due to not understanding the new PFL file layout, the client will receive an error, -EIO, without any other errors or crashes, when accessing a composite file. There is no way for these clients to be provided an alternate layout that would cover all the file data, so no server-mediated compatibility mode is possible without full IO redirection through some OSS node, which is not practical for performance reasons.

Lustre 2.10 clients that try and create a composite file on an older server, pre-2.10, through 'lfs setstripe' or through the modified API will receive an error, -EINVAL, without any other errors or crashes. Older servers will refuse to try and create a file with a PFL layout, due to the new magic value stored at the start of each layout.

Performance Testing

Performance testing will be used to verify the goals of the PFL feature. The performance goals of PFL are:

- Files written by few or only a single client should use one or only a few stripes to minimize metadata overhead and contention. Small PFL files should use one stripe.

- Files that are large or written by many clients concurrently should use as many stripes as possible to maximize bandwidth and distribute space usage across all OSTs. Large PFL files should stripe over all OSTs.

- Files with too many stripes will have high contention and poor metadata performance, so only the minimum number of stripes needed for each task should be created. PFL files should not use more stripes than needed for the anticipated workload.

For all performance testing, each thread/client will write 4GB of data to ensure enough data was written and read to avoid client-side caching effects due to RAM cache on the client nodes.

ID	Requirement/Test Description
P01	Single client 16 threads IOR read and write file per process with stripe = 1 and -1 and a composite file with a single component layout Purpose: This places all of the IO for each thread onto a single OST, and minimizes the amount of contention at each OST for small stripe width

	The stripe=1 results should show the optimum performance possible for applications writing in this mode. The composite file read and write numbers should be very close to the stripe=1 read and write performance.
P02	20 Clients 16 threads each IOR read and write file per process with stripe = 1 and -1 and a composite file with components over all OSTs <*Suggested layout?*> Purpose: confirm that minimizing the number of stripes provides best performance
P03	20 Clients 16 threads each IOR read and write single file with stripe = 1 and -1 and a composite file with components over all OSTs <*Suggested layout?*> Purpose: In order to get the maximum performance, the user or application would need to explicitly specify a higher stripe count. If the user is unaware of the need to explicitly specify a higher stripe count for the shared output file their application IO performance will be significantly below the possible peak value
P04	Test write performance across a PFL component boundry. Create a composite file with layout <*Suggested layout?*>, write to the file with small writes and increase write size until layout changes. Bandwidth per write will be recorded.

User Interface

The lfs command-line interface was extended to understand and manipulate composite files and their component layouts. lfs is the primary interface for users to create new files with a specific layout, get the layout of existing files, as well as setting default layout templates on directories that will be inherited by all new files and subdirectories created therein.

The lfs setstripe(1), lfs migrate(1), lfs getstripe(1), and lfs find(1) sub-commands will be extended to set and display the composite layout of a file, and to search for files with specific composite layout parameters or for

components that match specific parameters.

lfs setstripe

The lfs setstripe command creates a new file with the specified layout parameters, or sets the specified layout parameters as the default layout template on a parent directory.

Options added to 'lfs setstripe' to support PFL:

Flag	Description
--component-end | -E <comp_end>	comp_end is the extent end of the component. Can be specified with [+/-] k, M,G,T,P, or E (in KB, MB, GB, TB, PB, EB respectively, -1 for EOF), it must be aligned with the stripe_size
--component-del	Remove the component(s) specified by component ID or flags from an existing file. The ID specified by –I option is the numerical unique ID of the component, it can be obtained using the 'lfs getstripe' command.
--component-set	Isn't supported yet
--component-add	Add components to an existing composite file. The extent start of the first component to be added must be equal to the extent end of last component in the file. Must be used with the –E or –component-end flag.
--component-flags <comp_flags>	Specify certain type of components. Available flags: init - instantiated component ^init - non-instantiated component
--component-id | -I <comp_id>	The unique numerical component id

lfs find

The lfs find command is a Lustre-optimized and enhanced version of the find

command. The PFL specific flags added to 'lfs find' allow the user to find files that have components that match all flags specified.

Options added to 'lfs find':

Flag	Description
--component-end | -E <comp_end>	Find all files with a component with extent end comp_end. Can be specified with [+/-] k, M,G,T, or P (in KB, MB, GB, TB, PB respectively) comp_end must be aligned with the stripe_size
--component-count <comp_cnt>	Find all files with number of components comp_cnt in a composite file
--component-flags <comp_flags>	Find all files with component flag(s) comp_flags. Available flags: init - instantiated component.
--component-start <comp_start>	Find all files with a component with extent start comp_start. Can be specified with [+/-] k, M,G,T, or P (in KB, MB, GB, TB, PB respectively) comp_start must be aligned with the stripe_size

lfs getstripe

The lfs getstripe command prints some or all of the parameters of a file's layout. This is intended for regular users and administrators to query a particular file's layout, or the individual components of a composite file to examine the layout used to create the file. Without any of the option flags, a call to 'lfs getstripe' will display all the layout components.

Options added to 'lfs getstripe':

Flag	Description
--component-end= |-E<comp_end>	With no comp_end argument, returns the extent end of all components in the file. With an argument, returns all stripe information for that component.

	comp_end is the extent end of the component and can be used with [+/-] k, M,G,T,P, or E (in KB, MB, GB, TB, PB, EB respectively, -1 for EOF). comp_end must be aligned with the stripe_size
--component-count	Returns the number of components in the specified composite file
--component-flags=<comp_flags>	Specify certain type of components. Available flags: init - instantiated component ^init - non-instantiated component
--component-start=<comp_start>	With no comp_start argument, returns the starting byte extent of all components in the file. With an argument, returns all stripe information for that component with extent start comp_start. comp_start is the extent start of the component and can be used with [+/-] k, M,G,T,P, or E (in KB, MB, GB, TB, PB, EB respectively, -1 for EOF). comp_start must be aligned with the stripe_size
--component-id= | -I<comp_id>	With no argument, returns the unique numerical component id for all components in the file. With an argument, returns all stripe information for the component with comp_id

lfs migrate

The lfs migrate command moves (migrates) MDT inodes or OST objects between MDTs and OSTs, respectively.

Options added to 'lfs migrate' to migrate a regular file to a composite file, composite file to composite file, or composite file to regular file are:

Flag	Description
--component-end | -E <comp_end>	comp_end is the extent end of the component. Can

	be specified with [+/-] k, M,G,T, or P (in KB, MB, GB, TB, PB respectively, -1 for EOF), it must be aligned with the stripe_size

User Interface Testing

A new test suite has been created to exercise the user interfaces to manipulate composite files called sanity-pfl. The sanity-pfl test suite is composed of the following tests:

test 0 "Create full components file, no reused OSTs"
test 1 "Create full components file, reused OSTs"
test 2 "Add component to existing file"
test 3 "Delete component from existing file"
test 4 "Modify component flags in existing file"
test 5 "Inherit composite layout from parent directory"
test 6 "Migrate composite file"
test 7 "Add/Delete/Create composite file by non-privileged user"
test 8 "Run debench over composite files"
test 9 "Replay layout extend object instantiation"
test 10 "Inherit composite template from root"
test 11 "Verify component instantiation with write/truncate"
test 12 "Verify ost list specification"

The following user use case tests are to be run to ensure that the user interfaces work as expected.

ID	Requirement/Test Description	Test Suite	Test #
U01.1	User creates new file with fixed-size layout component using 'lfs setstripe –component-end'	sanity-pfl	0, 1
U01.2	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; --stripe-count or -c	sanity-pfl	0,2, 3,5, 6,7

U01.3	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; --stripe-size or -S	sanity-pfl	0,1,2, 6,8,9, 10
U01.4	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; --stripe-index or -i	sanity-pfl	5
U01.5	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; --pool or –p *Note: sanity-pfl test 0 and 1 should be modified to use pools*	*To Be Added*	N/A
U01.6	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; --ost-list or -o	sanity-pfl	1
U01.7	User creates new file with fixed-size layout component, passing an initial extent range for the first component in addition to existing setstripe parameters for this component; -d	sanity-pfl	5,10
U02.1	User adds component(s) with fixed-size extent(s) to an existing composite file	sanity-pfl	2
U02.2	User adds component(s) with fixed-size extent(s) to an existing composite file. The component ID will be assigned by the MDS automatically and will be unique for all components added on the file.	sanity-pfl	11 (patch 25717)
U03	User adds final component to existing composite file (using 'lfs –component-add')	sanity-pfl	2
U04.1	User requests the layout for an existing file. The new per-component field component ID will be printed for	sanity-pfl	0,1,5,11, 12

	each component.		
U04.2	User requests the layout for an existing file. The new per-component field extent range, start and end, will be printed for each component.	sanity-pfl	10
U04.3	User requests the layout for an existing file. The new per-component field flags will be printed for each component.	sanity-pfl	5
U04.4	User requests the layout for an existing file. The new per-component field FID will be printed for each component.	sanity-pfl	9
U04.5	User requests the layout for an existing file. The new per-component field component count will be printed for each component	sanity-pfl	2,3,5,6,9, 10
U05.1	User gets layout parameters to existing component by ID	sanity-pfl	0, 1, 5
U05.2	User gets specific layout parameter for stripe_count with component ID	*To Be Added*	
U05.3	User gets specific layout parameter for stripe_size with component ID	*To Be Added*	
U05.4	User gets specific layout parameter for pool with component ID	*To Be Added*	
U05.5	User gets layout parameters to existing component by "wildcard" component IDs such as LCME_ID_INIT	sanity-pfl	4 – test is always skipped
U05.6	User gets specific layout parameter for stripe_count with no component ID	*To Be Added*	
U05.7	User gets specific layout parameter for stripe_size with component ID	*To Be Added*	
U05.8	User gets specific layout parameter for pool with component ID	*To Be Added*	
U06.1	User accesses or modifies components in an existing	sanity-pfl	0,1,2,3, 6

	composite file; write operation		
U06.2	User accesses or modifies components in an existing composite file; read (dd) operation	sanity-pfl	2
U06.3	User accesses or modifies components in an existing composite file; truncate operation	sanity-pfl	11
U06.4	User accesses or modifies components in an existing composite file; chown operation shall properly assign ownership to all component OST objects of the file	*To Be Added*	
U06.5	User accesses or modifies components in an existing composite file; chgrp operation shall properly assign ownership to all component OST objects of the file	*To Be Added*	
U07	User deletes composite file with rm/unlink	sanity-pfl	3,5,6,7
U08	User creates a new composite file describing multiple components	sanity-pfl	0,1,3,5
U09	User migrates, with 'lfs migrate, composite file	sanity-pfl	6
U10	User searches, with 'lfs find', for composite files	*To Be Added*	
U11.1	User specifies default composite file template for directory	sanity-pfl	5,10
U11.2	User specifies default composite file template for directory and file created over rides default layout	*To Be Added*	
U12.1	Administrator specifies composite file template for root directory	sanity-pfl	10
U12.2	Administrator wants space balancing for large files in the filesystem (See Performance Testing)	*To Be Added*	
U13	User deletes uninitialized stripe component from file by ID with 'lfs setstripe –component-del	sanity-pfl	5,7
U14	User deletes uninitialized stripe component from file by flags	sanity-pfl	3
U15.1	Test bad input parameters; very large and negative values for –component-end	*To Be Added*	
U15.2	Test bad input parameters; add a component, with --	*To Be*	

	component-add, to an existing composite file where the last component was created with "-E-1". Test should fail.	*Added*	
U16	Test all 'lfs getstripe' composite file options; test '—component-start'.	*To Be Added*	

API

The llapi_layout_* interfaces provide an interface for userspace applications, including lfs, to specify plain and composite file layouts in an abstract manner, and then convert those abstract layouts into actual file layouts depending on the final attributes of the layout. For composite file layouts, the API will be extended to handle layouts with multiple components and other composite file specific attributes, for use in Lustre-specific tools such as lfs setstripe, lfs getstripe, and lfs find, as well as by end-user applications or libraries that want to create files with specific composite layouts to optimize file IO patterns, such as HDF5.

The llapi_layout API was extended to support composite layout with the following calls:

llapi_layout_comp_extent_get() - Get the extent, start and end offset, of the current layout component.

llapi_layout_comp_extent_set() - Set the extent, start and end, of a current component of the layout. The layout will be turned into composite if it was plain.

llapi_layout_comp_flags_get() - Layout component flags are used to indicate the status of the component, only LCME_FL_INIT is supported now, which indicates the OST objects of the component have been initialized. Get the attribute flags of the current component of a composite layout.

llapi_layout_comp_flags_set() - Layout component flags are used to indicate the status of the component, only LCME_FL_INIT is supported now, which indicates the OST objects of the component have been initialized. Set the specified flags of the current component and leave all other flags unchanged.

llapi_layout_comp_flags_clear() - Layout component flags are used to indicate the status of the component, only LCME_FL_INIT is supported now, which indicates the OST objects of the component have been initialized. Clear the specified flags of a composite layout and leave all other flags unchanged.

llapi_layout_comp_id_get() – Get the unique ID of the current layout component.

llapi_layout_comp_add() - Add one component to an existing layout. The layout will be turned into composite if it was plain before adding.

llapi_layout_comp_del() - Delete the current layout component from the composite layout. The component to be deleted must be the tail of layout component list, and it can't be the last one.

llapi_layout_comp_move_at() - Move the current component pointer of a layout file to a component with specified ID.

llapi_layout_comp_move() - Move the current component pointer of a layout file to a specified position: first component of the layout (LLAPI_LAYOUT_COMP_POS_FIRST), next position (LLAPI_LAYOUT_COMP_POS_ NEXT), or last component (LLAPI_LAYOUT_COMP_POS_LAST).

llapi_layout_file_comp_add() - Add one or more components to an existing composite file.

llapi_layout_file_comp_del() - Delete the specified layout component(s) from

an existing composite file. The unique ID must be specified or the ID is set to zero and a one of the following flags is specified:
LCME_FL_INIT – Instantiated components in the layout or
LCME_ID_NONE|LCME_FL_INIT - Non-instantiated components in the layout.

llapi_layout_file_comp_set() - Set certain attribute of the specified layout component(s). Not implemented.

For all API calls added for component layout, they return 0 on success, or -1 if an error occurred (in which case, errno is set appropriately). llapi_layout_comp_move() and llapi_layout_comp_move_at will return 0 on success, 1 when reaches last component when try to move next, or -1 if an error occurred (in which case, errno is set appropriately).

User Space API Use Cases

Several functions were added to the llapi_layout_* interfaces to accommodate the PFL feature were added to Lustre and must be exercised. The following are the requirements to exercise the new llapi_layout functions:

ID	Requirement/Test Description	Test Suite	Test number
A01	Application creates a new composite file with fixed-size component	sanity/llapi_layout_test	27D/31
A02	Application adds fixed-size component to an existing composite file	sanity/llapi_layout_test	27D/31
A03	Application adds last component to existing composite file - no extent is specified for an additional component	*To Be Added*	

A04	Application requests the entire layout for an existing composite file	sanity/llapi_layout_test	27D/30&31
A04.1	Application requests the entire layout for an existing composite file. The interface shall be able to efficiently manage composite files with at least as many components as supported by the underlying implementation.	*To Be Added*	
A05	Application selects an existing component by ID	sanity/llapi_layout_test	27D/31 – delete by ID
A06	Application accesses each component in an existing file	sanity/llapi_layout_test	27D/30&31 – POS_FIRST and POS_NEXT only. Missing POS_LAST
A06.1	Application accesses each component in an existing file. It shall be possible to select a subset of components using the attribute flags.	*To Be Added*	
A07	Application access attributes of a selected component without requiring them to understand the on-disk layout of the composite file or individual components	*To Be Added*	
A08	Application sets layout attributes of the selected component	sanity/llapi_layout_test	27D/30&31
A09	Application truncates composite file below component extent boundary. Once a layout component has been initialized, truncating down the file at an offset below	*To Be Added*	

	the start of the component will not affect the state of that or any later components. The OST objects allocated to the component will remain allocated.		
A10	Application writes beyond last defined file component. It is expected that uninitialized template layouts covering the whole file shall be specified at file creation time or inherited from the parent directory in their entirety.	*To Be Added*	
A11	Application reads beyond last defined file component and shall return a short read to the application, similar to reads beyond EOF for a plain file.	sanity-pfl	2
A12	Application creates composite file template with multiple components. The llapi_layout interface shall allow multiple components with different layout parameters to be specified before the file is created. The MDS will create the file with an uninitialized layout template, and only initialize layout components as they are written to.	sanity-pfl	11
A13	Application writes within an uninitialized file component. Writing to a file offset within an uninitialized file component shall cause the client to indicate to the MDS that it needs to initialize the component(s) within that extent, and the MDS shall allocate OST objects for the affected component(s) and store them into the	sanity-pfl	11

	layout. This shall cause layout revocation from any other clients accessing the same file and cause them to refresh their layout (if they are still accessing the file) so that they can see the changes to the affected component(s).		
A14	Application reads from an uninitialized file component. Reading from a file offset within an uninitialized file component shall return no data, as there will not be any OST objects to read	sanity-pfl	2
A14.1	Application reads from an uninitialized file component. Reading from a file offset within the file size, the client will zero-fill the buffers and return an appropriate number of bytes to the caller.	sanity-pfl	2
A15	Application truncates beyond last initialized file component. File size shall be as expected.	sanity-pfl	11
A16	Multiple clients write beyond last initialized file component. If multiple clients request the MDS initialize the same component concurrently, the MDS will serialize these requests with the layout lock of the file and initialize all of the requested components.	*To Be Added*	
A16.1	Multiple clients write beyond last initialized file component. If multiple clients request the MDS initialize different components concurrently, the MDS will serialize these requests with the layout lock of the file	*To Be Added*	

	and initialize all of the requested components.		
A17	Application deletes selected uninitialized stripe component. This shall only be allowed for the last component of a file, and only if that component is uninitialized (i.e. no OST objects are currently allocated to the component), as this may otherwise cause data loss.	sanity/llapi_layout_test	27D/30
A18.1	Unexpected input values: llapi_layout_comp_move() - input values > 2 and < 0 for pos	*To Be Added*	
A18.2	Unexpected input values: llapi_layout_file_comp_del() use NEXT after the last component	*To Be Added*	
A19	llapi_layout_file_comp_set() is not implemented. Test that it returns EOPNOTSUPP	*To Be Added*	
A20	Exercise all API calls. The following are not used in llapi_layout_test.c: llapi_layout_comp_flags_get/set/clear(), llapi_layout_comp_move_at(), llapi_layout_file_comp_set()	*To Be Added*	

Feature Interaction

HSM

A composite file must be archived, releases and restored. The expectation is that a composite file we be restored with the same number of components and

extent ranges as the archived file. There is no requirements on the OST indexes of the archived and restored composite files.

LFSCK

LFSCK consistency checking and repair for the compound layouts will detect and correct filesystem corruption or implementation defects. The LFSCK tool shall correctly process files with a composite layout, verifying all OST objects referenced by the layout exist, and have a "fid" xattr that correctly references the component and sub-layout index in which the object appears.

Ost-pools

One of the premises of PFL is that it is possible to specify different pools for each component to control on which OSTs those components are located.

Snapshot

A snapshot shall be taken of a file system with composite files in it and where the file system has a default composite file layout specified. The snapshot shall preserve the composite file layout if the snapshot is mounted and files are retrieved.

Subdirectory Mount

Assume there is a default composite layout specified at the file system level and there are different composite file layouts specified at every level of several subdirectories. When a subdirectory is mounted on a client and files are created in the sub directory, the files will inherit the composite file layout of the parent directory, if a default layout exists, or will inherit the file system default layout template if it exists.

Distributed Name Space (DNE)

Composite files will be created under remote directories and striped directories.

ID	Requirement/Test Description	Test Suite	Test #

F01	HSM: Archive, release and restore a composite file.	sanity-hsm	1b
F02	HSM: Archive and restore a composite file. The layout after HSM restore should be the same as the original file (meaning same extent range [start, end] for every component, but not the same OST id of every component stripe objects)	*Missing* – sanity-hsm test_1b does not check matching number of components and extent.	
F03	LFSCK: LFSCK shall check and repair PFL layouts	sanity-lfsck	13,14, 15a,15b, 17,18a-e,18h
F04	SNAPSHOT: Snapshots preserve component file layouts	*To Be Added*	
F05	SUBMOUNT: Files created under a subdirectory with a default composite file layout will inherit the file system default layout.	*To Be Added*	
F06	DNE: Composite files will be created under remote directories	*To Be Added*	
F07	DNE: Composite files will be created under striped directories.	*To Be Added*	

What Not to Test

There is no aspect of the PFL feature that should not be tested.

Results

Results will be included when they are available.