PVFS2: System Interface Test Suite

Frank Shorter
Michael Speth

May 16, 2003

1 Introduction

The Parallel Virtual File System version 2 (PVFS2) is in development. This document attempts to outline the process of validating the System Interface of the client. The testing process is divided into two parts: positive tests and negative tests. Positive tests examine functions operating on normal procedures. Negative tests examine the functions behavior on abnormal procedures designed to make the functions fail.

2 Setup

Describe the system (hardware and OS) that the tests are being run on and the version of pvfs2.

3 Positive Tests

The positive tests will verify basic functionality, and ensure that the system interface behaves as expected for a given set of reasonable inputs. We expect that all of these calls should succeed. It is the goal of this section to provide coverage for all areas of the system interface that will recieve the most usage.

3.1 Startup and Shutdown

The most trivial test of the system interface, we initialize and finalize the system interface.

3.2 Metadata tests

1. **File creation**: We will test the creation of files with valid attributes inside of directories where we have permission to do so. The number of data files will be varied from 1 to 2N (where N is the number of I/O servers). Creation will be verified with a lookup operation.

2. **File removal**: We will test the removal of files that have the appropriate permissions for our user. Removal will be verified by a failed lookup operation. After the file is removed, we will re-create a new file with the same name. Lookup of the new file must return the new handle. as well as the attributes of the new file(including datafile handles).

3. **Setting/retrieving attributes on a file**: Setting/retrieving attributes will be tested by setting all the attributes on a file to some known values, then calling getattr to ensure that they have been set. Important things to pay attention to here are the filesize, as well as permissions.

4. **Lookup of a file**: Lookup will be tested by creating a file, and then looking up that file and comparing the handles. Create and Lookup should return the same handles and file system id numbers.

5. **Renaming files**: We will create a file, lookup the file, then call rename. We will verify rename by calling lookup on the old filename, ensuring that it fails, and then calling lookup on the new filename and ensuring that it returns the handle we were given at create time. Renaming will need to be tested within the same directory, as well as renaming(essentially moving) files into different directories.

6. **Directory creation**: We will test the creation of directories with valid attributes inside of directories where we have permission to do so. We're only looking to create a moderate number of directories with this test case. Please refer to the stress testing section for info on the directory tests where a very large number of directories are added. directory creation will be verified both by having it appear when readdir is called, as well as being able to look it up with the lookup function.

7. **Directory removal**: We will create a directory, verify that it exists with readdir and lookup, then call rmdir. To ensure that it has been deleted, we will attempt to lookup and call readdir on the directory name that was just removed. Both of these calls must not return any trace of the directory. Additionally, we will create a directory of the same name, and compare its attributes to the previous directory.

8. **Lookup of a directory**: Lookup will be tested by creating a directory, and then looking up that directory and comparing the handles. Create and Lookup should return the same handles and file system id numbers.

9. **Setting/retrieving attributes on a directory**: Setting/retrieving attributes will be tested by setting all the attributes on a directory to some known values, then calling getattr to ensure that they have been set.

3.3 I/O tests

3.3.1 Reading

The read tests will be performed on files where we have read permission, and data exists within the file. The request will be committed prior to the IO call. We will need to test every combination of requests (from pvfs-request.h):

1. **Contiguous**: count should be varied to ensure that we're hitting multiple servers as well as only getting data from each server at time.
2. **Vector**: with "stride" lengths that span multiple servers.
3. **Hvector**: see vector.
4. **Indexed**: indexed will be tested with varying block lengths and displacements. displacements that cause multiple servers to be spanned as well as large block lengths will also be used.
5. **Hindexed**: see indexed.
6. **Manual ub/lb/extents**: Calling read with varying the displacements on ub, lb, and extents will be performed.

3.3.2 Writing

The write tests will be performed on files where we have write permission. Data may or may not exist within file prior to calling write. The request will be committed prior to the IO call. We will need to test every combination of requests (from pvfs-request.h):

1. **Contiguous**: count should be varied to ensure that we're hitting multiple servers as well as only getting data from each server at time.
2. **Vector**: with "stride" lengths that span multiple servers.
3. **Hvector**: see vector.
4. **Indexed**: indexed will be tested with varying block lengths and displacements. displacements that cause multiple servers to be spanned as well as large block lengths will also be used.
5. **Hindexed**: see indexed.
6. **Manual ub/lbextent**: Calling write with varying the displacements on ub, lb, and extents will be performed.

3.3.3 Truncate

The truncate tests will be performed on files where we have write permission. This test will need to be performed on files of size 0, as well as files with data on every combination of servers.

4 Negative Tests

The negative tests are broken up into two sections: invalid parameters and functional ordering. The invalid parameters tests examines the functions' behaviors when invalid parameters are supplied. The tests for functional ordering examines functions' behaviors when the ordering of functions are incorrect.

4.1 Invalid Parameters

Tests functions' behavior when invalid parameters are supplied

4.1.1 Null parameters

All parameters of each function are null. Note, before any of the functions can be called, initialize must be called with valid parameters expect for tests regarding initialize.

1. Call initialize and set its parameters to null. Record the return value and error code returned.
2. Call finalize and set its parameters to null. Record the return value and error code returned.
3. Call lookup and set its parameters to null. Record the return value and error code returned.
4. Call getattr and set its parameters to null. Record the return value and error code returned.
5. Call setattr and set its parameters to null. Record the return value and error code returned.
6. Call mkdir and set its parameters to null. Record the return value and error code returned.
7. Call readdir and set its parameters to null. Record the return value and error code returned.
8. Call create and set its parameters to null. Record the return value and error code returned.
9. Call remove and set its parameters to null. Record the return value and error code returned.
10. Call rename and set its parameters to null. Record the return value and error code returned.

11. Call symlink and set its parameters to null. Record the return value and error code returned.
12. Call readlink and set its parameters to null. Record the return value and error code returned.
13. Call read and set its parameters to null. Record the return value and error code returned.
14. Call write and set its parameters to null. Record the return value and error code returned.

4.1.2 Varied Null Parameters

Some of the parameters of each function are null. Note, before any of the functions can be called, initialize must be called with valid parameters expect for tests regarding initialize.

1. Iterate through the list found in section 4.1.1 with the first parameter set to null. The remaining parameters (if there are any) are set to a valid value. Record the return value and error code returned.
2. Iterate through the list found in section 4.1.1 with the second parameter (if there is one) set to null. The remaining parameters (if there are any) are set to a valid value. Record the return value and error code returned.
3. Iterate through the list found in section 4.1.1 with the third parameter (if there is one) set to null. The remaining parameters (if there are any) are set to a valid value. Record the return value and error code returned.
4. Iterate through the list found in section 4.1.1 with the first and second parameter (if there is one) set to null. The remaining parameters (if there are any) are set to a valid value. Record the return value and error code returned.

4.1.3 Invalid File

All test cases use an invalid file

1. Call lookup and set the pinode_refernce.handle to -1. Record the return value and error code returned.
2. Call getattr and set its parameters to null. Record the return value and error code returned.
3. Call setattr and set its parameters to null. Record the return value and error code returned.
4. Call mkdir and set its parameters to null. Record the return value and error code returned.
5. Call readdir and set its parameters to null. Record the return value and error code returned.

6. Call create and set its parameters to null. Record the return value and error code returned.
7. Call remove and set its parameters to null. Record the return value and error code returned.
8. Call rename and set its parameters to null. Record the return value and error code returned.
9. Call symlink and set its parameters to null. Record the return value and error code returned.
10. Call readlink and set its parameters to null. Record the return value and error code returned.
11. Call read and set its parameters to null. Record the return value and error code returned.
12. Call write and set its parameters to null. Record the return value and error code returned.

4.2 Functional Ordering

All test cases use the pre-built file found in section 2.

4.2.1 Client uninitialized

Test the behavior of all functions when the initialize function has not been called.

1. Call lookup and record the return value and error codes.
2. Call getattr and record the return value and error codes.
3. Call setattr and record the return value and error codes.
4. Call mkdir and record the return value and error codes.
5. Call readdir and record the return value and error codes.
6. Call create and record the return value and error codes.
7. Call remove and record the return value and error codes.
8. Call rename and record the return value and error codes.
9. Call symlink and record the return value and error codes.
10. Call readlink and record the return value and error codes.
11. Call read and record the return value and error codes.
12. Call write and record the return value and error codes.

4.2.2 Client unfinalized

Test the behavior of the system when the system is initialized but the program exits without calling finalize. Another program is run after the previous program exited and all functions are tested including initialize and finalize.

1. Call initialize and exit the program. Call initialize and record the return value and error codes.
2. Call initialize and exit the program. Call initialize then lookup and record the return value and error codes.
3. Call initialize and exit the program. Call initialize then getattr and record the return value and error codes.
4. Call initialize and exit the program. Call initialize then setattr and record the return value and error codes.
5. Call initialize and exit the program. Call initialize then mkdir and record the return value and error codes.
6. Call initialize and exit the program. Call initialize then readdir and record the return value and error codes.
7. Call initialize and exit the program. Call initialize then create and record the return value and error codes.
8. Call initialize and exit the program. Call initialize then remove and record the return value and error codes.
9. Call initialize and exit the program. Call initialize then rename and record the return value and error codes.
10. Call initialize and exit the program. Call initialize then symlink and record the return value and error codes.
11. Call initialize and exit the program. Call initialize then readlink and record the return value and error codes.
12. Call initialize and exit the program. Call initialize then read and record the return value and error codes.
13. Call initialize and exit the program. Call initialize then write and record the return value and error codes.

4.2.3 Client finalized

The initialize function is called and immediately after the finalize function is called. Test behavior of system functions under this scenario.

1. Call initialize then finalize. Immediately after finalize, call lookup and record the return and error codes.
2. Call initialize then finalize. Immediately after finalize, call getattr and record the return and error codes.
3. Call initialize then finalize. Immediately after finalize, call setattr and record the return and error codes.
4. Call initialize then finalize. Immediately after finalize, call mkdir and record the return and error codes.
5. Call initialize then finalize. Immediately after finalize, call readdir and record the return and error codes.
6. Call initialize then finalize. Immediately after finalize, call create and record the return and error codes.
7. Call initialize then finalize. Immediately after finalize, call remove and record the return and error codes.
8. Call initialize then finalize. Immediately after finalize, call rename and record the return and error codes.
9. Call initialize then finalize. Immediately after finalize, call symlink and record the return and error codes.
10. Call initialize then finalize. Immediately after finalize, call readlink and record the return and error codes.
11. Call initialize then finalize. Immediately after finalize, call read and record the return and error codes.
12. Call initialize then finalize. Immediately after finalize, call write and record the return and error codes.

4.2.4 Operations on non-existent Files

Tests for functions that operate on existing files on a file that has not been created.

1. Call initialize then lookup on a file that has not been created. Record the return value and error codes.
2. Call initialize then getattr on a file that has not been created. Record the return value and error codes.
3. Call initialize then setattr on a file that has not been created. Record the return value and error codes.
4. Call initialize then readdir on a file that has not been created. Record the return value and error codes.
5. Call initialize then remove on a file that has not been created. Record the return value and error codes.
6. Call initialize then rename on a file that has not been created. Record the return value and error codes.
7. Call initialize then symlink on a file that has not been created. Record the return value and error codes.
8. Call initialize then readlink on a file that has not been created. Record the return value and error codes.
9. Call initialize then read on a file that has not been created. Record the return value and error codes.
10. Call initialize then write on a file that has not been created. Record the return value and error codes.

4.2.5 Repeated Operations: meta data

Continually call functions that change the meta data of a file.

1. Call initialize and then call setattr on the same test file 100 times. Record return values and error codes.
2. Call initialize and then call rename 100 times. Record return values and error codes.
3. Call initialize and then call symlink 100 times. Record return values and error codes.

4.2.6 Repeated Operations: create

Continually call functions on one file that create new files such as mkdir and create.

1. Call initialize and then call mkdir on the same test file 100 times. Record return values and error codes.
2. Call initialize and then call create on the same test file 100 times. Record return values and error codes.
3. Call initialize and then call mkdir on differnt test files 100 times. Record return values and error codes.
4. Call initialize and then call create on differnt test file 100 times. Record return values and error codes.

5 Results