

# **Design and Development of Virtual Volume File System**

**A CS1980 Capstone Project**

James D. McMillian

December 13, 2013

## Introduction

As a student pursuing the completion of the Capstone Project, at The Pittsburgh Supercomputing Center (PSC), National Resource for Biomedical Supercomputing, under the guidance of Art Wetzel, I was charged with the design, development and testing of an Active Virtual File System for the manipulation of massive electron microscopy datasets required for connectomics research.

The goal of the project was to develop and test a prototype virtual file system following the concept presented by Professor Wetzels and his colleagues at the Janelia Farm conference in Oct 2012[1] which would aid in the handling, and curation of petabyte scale datasets. A previous attempt to develop a prototype file system had uncovered a short coming in adapting an existing tool, ScriptFS, to meet key operational requirements.

This semester long project ultimately was an exploration to develop an alternative implementation that could be used for this intended purpose. Development and system tests were performed on an Ubuntu 12.04 Linux operating system utilizing the FUSE module file system. Compilation of the projects svfs.cpp code was using g++ with the following command line:

```
g++ -Wall -Wno-sign-compare src/svfs.cpp `pkg-config fuse --cflags --libs` -o vvfs
```

An overview of C and C++, the Linux operating system, and more specifically the Ubuntu 12.04 distribution of Linux, and a detailed understanding of the FUSE file system will be explored in more detail.

## Motivation and Goals

Wetzel and colleagues at the PSC are currently working on a framework to handle the big-data being generated from electron microscopy of a mouse brain at the nanometer scale. They are currently working on two datasets of 100TB each, and expected to reach Petabytes in 2-3 years [2].

Each dataset consists of thousands to millions of image tiles. They form stacks of thin cut brain tissue. The largest stack contains approximately 10,400 sections totaling approximately 109.05TB in size. Each section is composed of a 4 by 4 grid totaling 16 tiles, where each tile is electron microscopy scan of 25600 by 25600 pixels at 4nm per pixel and a slice thickness of 30nm. Each pixel is an 8bit value representing the intensity of electron scattering at that point. The scans were taken from tissue of mouse brain collected and imaged at Harvard in Dr. Jeff Lichtman's laboratory.

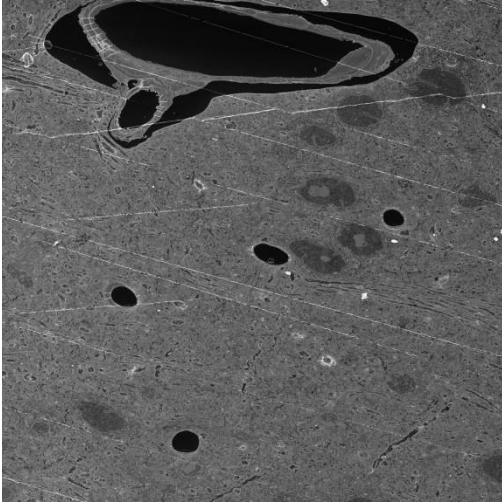


Figure 1. Inverted image of S4S4r3-c4.jpg

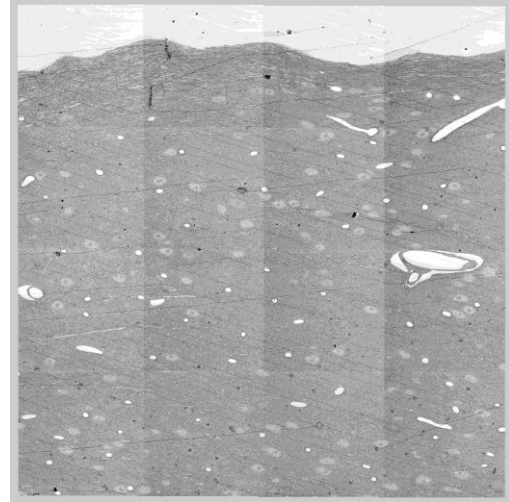


Figure 2. Composited image of section S8eoS2S4\_100.jpg.

The raw data has multiple artifacts and wrinkles created from the slicing and scanning process (see figure 1), in addition to the inconsistencies between sections such as tonal variation and alignment issues (see figure 2 for tonal variation issues).

Therefore, a need arises to modify the data in such a manner as to minimize these inconsistencies without the drawback of duplicating the datasets. Such a technique would need to perform on the fly transformations of the data on an on-demand basis, and in such a manner that would hinder neither the end users, nor the hosting file servers' performance to any significant degree.

### Background and previous work

In the spring 2013 semester, David Stein attempted to modify the already existing ScriptFS file system to suit the needs of this work. The process resulted in determining that while ScriptFS could be forced into many useable scenarios for the intended goal, ultimately the large number of script files required to fully satisfy the requirements would make the ScriptFS solution unworkable in practice [3].

The Primary shortcomings of ScriptFS lies in its inability to have a script perform more than a single variation of its designed intent. In other words, if a script were to add two numbers, then a script for each combination of numbers would need to be created. For example, if the Script name was ADD, and the legal values to add consisted of 0, and 1 only. Then 4 scripts would be required. ADD0+0, ADD0+1, ADD1+0 and ADD1+1. Taking this a reasonable step further, if 3 digits ranging from 0 to 999 were to be allowed, then in order to utilize ScriptFS in this aspect, 1,000,000,000 script files would need to be generated. This is obviously not an acceptable solution. Attempts to add more flexible parameter handling were not successful. Contact with the ScriptFS author confirmed that such an effort would be a complete rewrite. And thus, ScriptFS failed to become an acceptable long term solution. However while ScriptFS had potential, it ultimately could not be utilized since the issues were fully embedded within its code base. [3] Additionally, ScriptFS has issues handling data larger than 64KB in size.

A review of ScriptFS however revealed its code based resided within the FUSE (Filesystem in Userspace) codebase. A review of the FUSE website at [sourceforge.net](http://sourceforge.net), revealed a number of file system based on FUSE. Ultimately, none of the, file system available through FUSE's website were workable for the

desired solution, but due to the sheer magnitude of variations in filesystems based on the FUSE code base, close inspection of FUSE became warranted and experimentation with the FUSE code base began.

The initial application developed was the FUSE HelloWorld example [4]. This illustrated that a basic file system could be implemented where the file system itself could generate output that external programs could access as if the target files actually existed on the hard drive. Experiments were then performed to output the results of varying directory names passed into the new virtual file system, and eventually arguments and parsing were determined to be viable for the path being used within the virtual file system.

The initial proof of concept was to a simple ADD function as a directory entry, with a single file contained within that directory name EVALUATE.TXT. The ADD directory would have 2 numbers contained within the path separated by a comma, and encapsulated within parenthesis. Such a command would be represented as:

```
../ADD(5,7)/EVALUATE.TXT
```

When the file was read using the Linux CAT command or gnomes text editor GEDIT, the file would return the text:

```
The answer is 12.
```

Since the custom virtual file system developed parsed the directory path, any number of arguments could be passed into the system in the manner. So that arguments could be of any length of parameters as necessary.

### Algorithms developed and implemented

With a working proof of concept virtual file system. A proposal was presented to Dr. Wetzel and his colleagues to pursue development of a prototype virtual file system that could produce transformations of actual data files on the fly. It was determined that this technique could bypass the limitations imposed by ScriptFS.

A requirement for a working file system was determined to need 3 directories: 1. A source directory that the actual file would reside that could be manipulated, 2. A directory to mount the virtual file system at, and 3. A directory that would contained various transformations to be loaded upon instantiation of the volume virtual file system (vvfs). An example mount command would look like this:

```
./vvfs mount=/home/username/vvfs/mount data=/home/username/vvfs/data  
functions=/home/username/vvfs/extensions
```

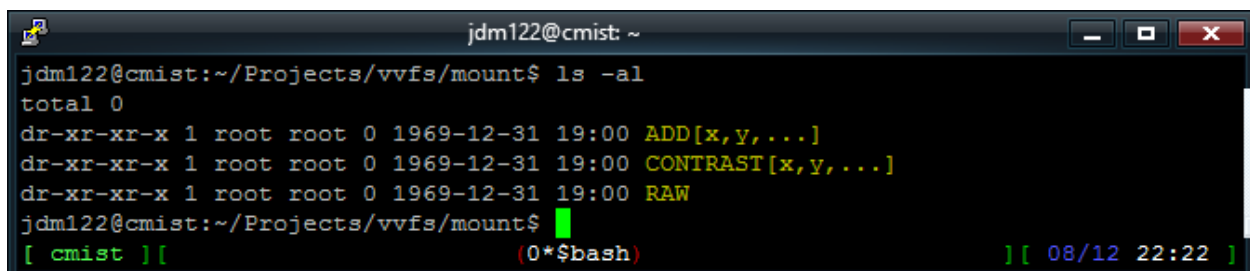
The initial transform operation was constrained to be a relatively small set of operations, primarily focused on establishing the ability to return large pieces of modified data. A RAW command and a CONTRAST command were eventually determined to be the most effective pair of operations to implement this early in the design stage.

The RAW command would in the simplest manner read the data from the file located at the end of the VVFS command path, and return the original data to the program accessing the VVFS. While this resulted in zero data modification, it proved to be a simple test to determine if that VVFS could adequately transfer large amounts of data through the file system without error. After several trial runs and testing, the code base was updated to handle large file sizes, those greater than 32bits in file size. The new code was specifically written to handle 64bit file length sizes. Which resolved all large file size issues previous experienced using ScriptFS.

The CONTRAST command was then implemented by reusing the code for the RAW command, and applying a mathematical equation to it. The equation  $Y = C_n X^n + C_{n-1} X^{n-1} + \dots + C_1 X^1 + C_0$  Where Y is the new value being passed to the calling program, and X is the value from the original source file. And  $C_n$  is a constant being passed into the equation from the CONTRAST operation as a parameter.

**../CONTRAST[0.0004, 0,255]/targetimage.pgm**

The file system then had to be expanded to handle multiple basic operations such as directory listing and directory list properties. Otherwise many of the basic operations would fail to function properly. In the end, the root directory of the VVFS contained 3 basic operations, ADD[x,y,...], CONTRAST[x,y,...] and RAW as seen in figure 3, the mounted and executing VVFS.



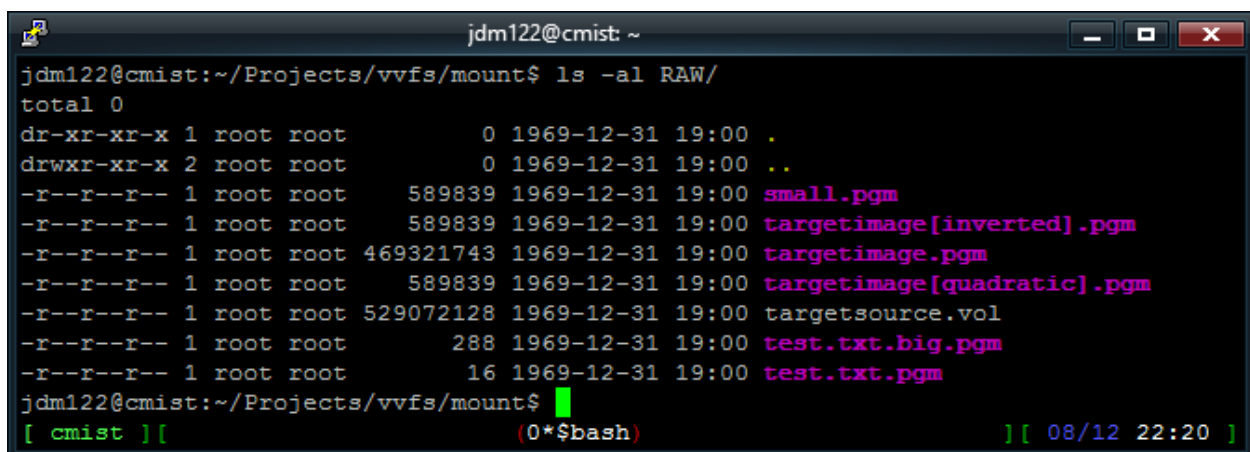
```

jdm122@cmist: ~/Projects/vvfs/mount$ ls -al
total 0
dr-xr-xr-x 1 root root 0 1969-12-31 19:00 ADD[x,y,...]
dr-xr-xr-x 1 root root 0 1969-12-31 19:00 CONTRAST[x,y,...]
dr-xr-xr-x 1 root root 0 1969-12-31 19:00 RAW
jdm122@cmist:~/Projects/vvfs/mount$
[ cmist ] [(0*$bash) ] [ 08/12 22:22 ]

```

Figure 3. Root directory of VVFS

Examining the contents of the CONTRAST and RAW directories revealed the contents of the source directory, see figures 4 and 5.



```

jdm122@cmist: ~/Projects/vvfs/mount$ ls -al RAW/
total 0
dr-xr-xr-x 1 root root 0 1969-12-31 19:00 .
drwxr-xr-x 2 root root 0 1969-12-31 19:00 ..
-r--r--r-- 1 root root 589839 1969-12-31 19:00 small.pgm
-r--r--r-- 1 root root 589839 1969-12-31 19:00 targetimage[inverted].pgm
-r--r--r-- 1 root root 469321743 1969-12-31 19:00 targetimage.pgm
-r--r--r-- 1 root root 589839 1969-12-31 19:00 targetimage[quadratic].pgm
-r--r--r-- 1 root root 529072128 1969-12-31 19:00 targetsource.vol
-r--r--r-- 1 root root 288 1969-12-31 19:00 test.txt.big.pgm
-r--r--r-- 1 root root 16 1969-12-31 19:00 test.txt.pgm
jdm122@cmist:~/Projects/vvfs/mount$
[ cmist ] [(0*$bash) ] [ 08/12 22:20 ]

```

Figure 4. Result of issuing the command "ls -al RAW/"

```

jdm122@cmist: ~
jdm122@cmist:~/Projects/vvfs/mount$ ls -al CONTRAST\[x\,y\,...\]/
total 0
dr-xr-xr-x 1 root root      0 1969-12-31 19:00 .
drwxr-xr-x 2 root root      0 1969-12-31 19:00 ..
-r--r--r-- 1 root root 589839 1969-12-31 19:00 small.pgm
-r--r--r-- 1 root root 589839 1969-12-31 19:00 targetimage[inverted].pgm
-r--r--r-- 1 root root 469321743 1969-12-31 19:00 targetimage.pgm
-r--r--r-- 1 root root 589839 1969-12-31 19:00 targetimage[quadratic].pgm
-r--r--r-- 1 root root 529072128 1969-12-31 19:00 targetsource.vol
-r--r--r-- 1 root root 288 1969-12-31 19:00 test.txt.big.pgm
-r--r--r-- 1 root root 16 1969-12-31 19:00 test.txt.pgm
jdm122@cmist:~/Projects/vvfs/mount$
[ cmist ] (0*$bash) [ 08/12 22:22 ]

```

Figure 5. Result of issuing the command “ls -al CONTRAST[x,y,...]/”

Additionally, exploring the data generated from those directories resulted in the dynamic manipulation of the data from the source files as the data was requested by the calling applications. The test data images of a CAT scan of a mouse using the RAW and CONTRAST directories resulted in properly manipulated images as seen in Figures 7, 8 and 9. Figure 7 represents the original data, where the data is being passed through the RAW operation, thus the seen image, is identical to the image stored on the backend file server. While figure 8 and 9 show manipulated data according to two different sets of parameters passed into the same operation to adjust the contrast levels of the image as it passes through the VVFS. Each contrast curves can been in Figure 6.

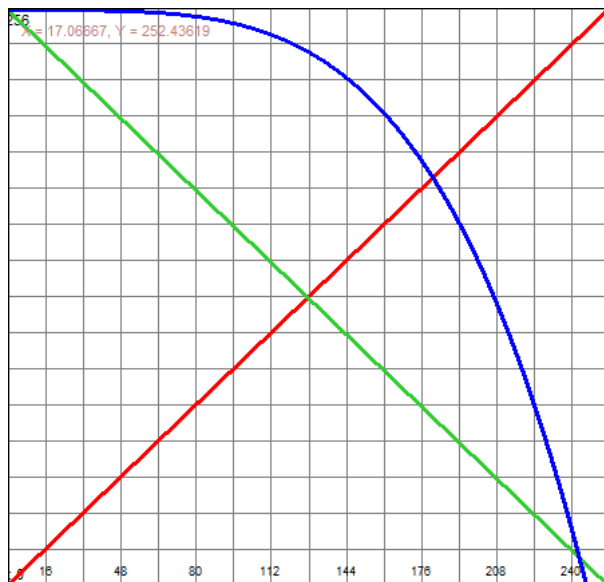


Figure 6. Contrast curves applied to target image.

Red:  $Y = X$     Green:  $Y = -1(X^1) + 255(X^0)$   
Blue:  $Y = -0.00000007 * (X^4) + 255(X^0)$

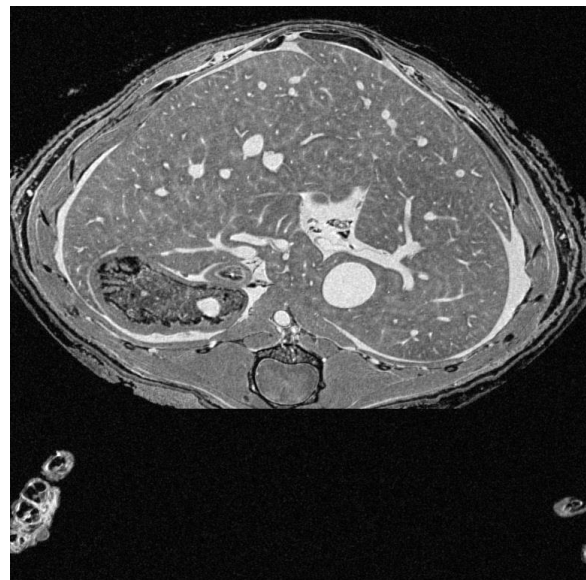


Figure 7: Result of contrast using the red curve  
 $Y = X$



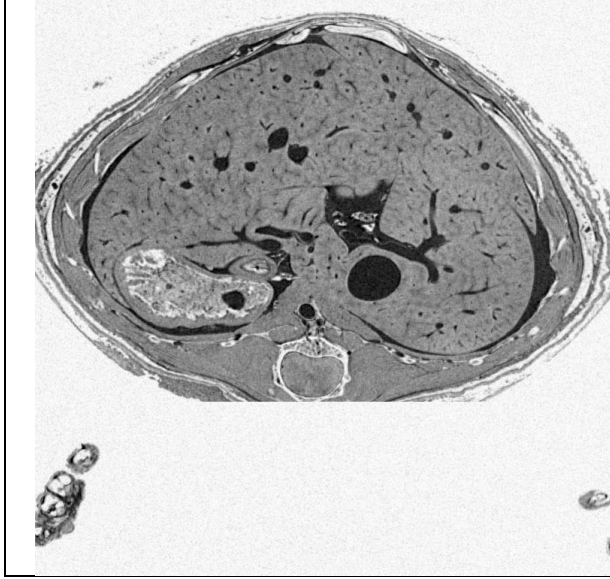


Figure 8: Result of contrast using the green curve  
 $Y = -1(X^1) + 255(X^0)$

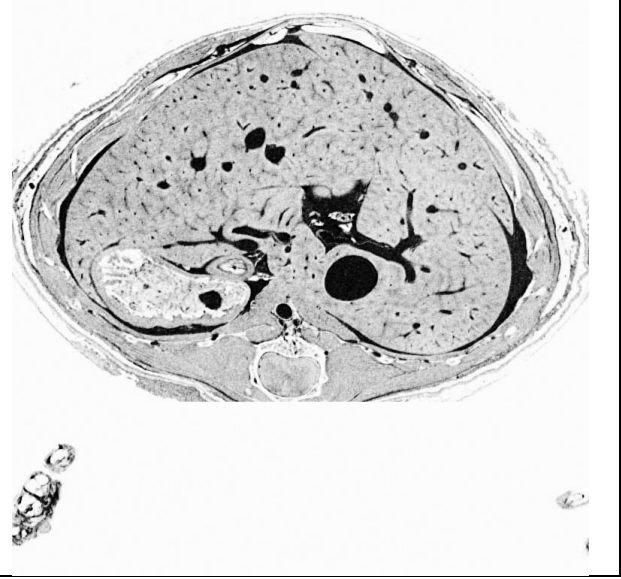


Figure 9: Result of contrast using the blue curve  
 $Y = -0.00000007 * (X^4) + 255(X^0)$

### Evaluation and analysis of algorithms implemented

Finally speed tests needed to be performed to determine the transformation would significantly impact the operations of the system. The number of clock ticks was calculated before and after the execution of each command, and the elapsed time between start of finish during the many read requests were recorded. The data was evaluated shows that no significant difference in the read operations between the various transformations was experienced. Although the data from the CONTRAST[0.0004,0,255] varied wildly as compared to CONTRAST[-0.0004,0,.255], see figure 10. This could have been caused from any number of external factors ranging from multiple file access to the source file, to caching concerns performed by the operating system itself. Ultimately, the results of the speed analysis are inconclusive, and further testing will need to be developed and executed to be able to definitively state one way or another if the system will become a feasible solution to the big data problem.

Additional tests should be performed to compare the VVFS system access against actual real file access using the true file system in order to determine the impact the VVFS will have on the system performance. Such tests are in design as of the writing of this paper, but have as of yet to be implemented.

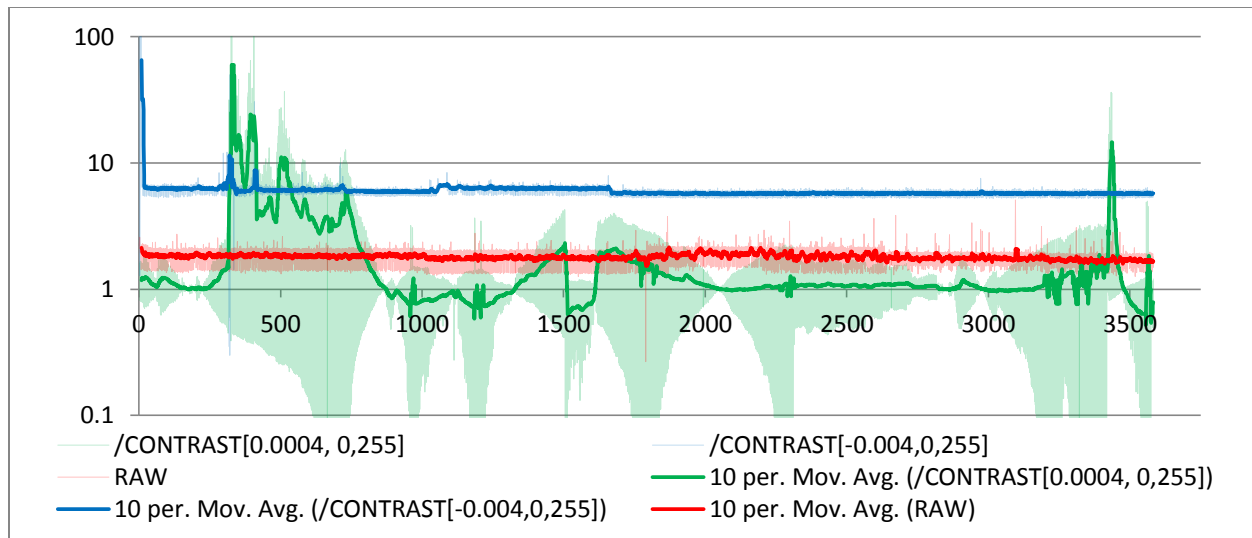


Figure 10. Ticks per Byte throughput of Multiple Virtual Transforms

## Summary

The use of FUSE to develop the VVFS appears to hold promise as a means to accomplish the goals of this project. By being able to pass in the arguments necessary to perform the transformations, a single transformation module can be developed to perform a wide array of variations upon the target data without the overhead of multiple files, one for each transform of the same time. This was the major drawback generated by ScriptFS. As a result redundant data is not only reduced to a minimal size, but eliminated in its entirety, the exception being if some form of directive structure ends up being required. However such directive structure has not been explored, nor discussed in this paper as it currently resides beyond the scope of the project. The fundamental framework for the VVFS has been established, and initial tests have been executed. All resulted in informative results, even though more testing clearly needs to be performed.

While it is unclear if the VVFS through the use of FUSE will be efficient enough after further testing. One thing is for certain, the use of FUSE to develop the VVFS holds promise, and further testing is definitely warranted based solely on its ability to satisfy all the requirements as are currently known.

## Future development

Future development should be focused on determining the exact impact FUSE will have upon the normal file system and what degradation of performance will occur through the use of FUSE. It is almost certain that some perform drop will occur, so the question should not be whether there will be a performance drop, but by how much will it be, and how much is considered to be unacceptable.

Additionally, the current method of logging statistical data has an adverse impact on the system performance and would need to be modified in order to minimize its performance impact on the system during the testing cycles. Additionally, event logging would likewise need to be modified so as to minimize system impact during testing and normal execution time.



## Reflection on the project

On average about 10 hours per week were spent on the project, either writing and debugging code, or becoming familiar with the FUSE api. At the beginning of the semester the FUSE api was unavailable, and exploration was, for the most part, a matter of trial and error. Testing different parameter options to cryptic operations became the only viable means to determine their functionality. It was only near the end of the semester that such operations as when the user closes a file, FUSE passes execution to a callback command name 'release' which was initially undocumented. Such occurrences quickly became normal during the course of the semester. Once the API was put back online by the developers of FUSE, many of the previously cryptic commands and system became clear. A deeper understanding of the inner workings of FUSE would have greatly increased this developer's ability to efficiently write code for the new VVFS.

My mentor Art was of immense benefit in understanding the previous attempts to implement ScriptFS, and in understanding the depth of the need for the VVFS. Additionally he was readily available either in his office or via email.

My personal thoughts on the project are that I would like to continue working on it, and in cooperation with the PSC, and with Art, I plan on continuing to develop the VVFS into the spring semester as a volunteer so long as I am able to. With a better understanding of the FUSE file system, development and design of a VVFS should move smoother and quicker now.

As a final thought, the experience has shown me that my real passion lies in the ability to develop new ideas to solve problems, and it's was my intention before this semester to continue into graduate school. Now after having spent some time working with professors of academia, it remains my intention to continue into graduate school to work on similar problems.

## References

[1] A. Wetzel, G. Hood, M. Dittrich (2012, October 28) Facilitating large scale data analysis using an active filesystem computing framework [Online]. Available:  
[http://staff.psc.edu/awetzel/jan\\_imgs2know.pdf](http://staff.psc.edu/awetzel/jan_imgs2know.pdf)

[2] A. Wetzel, (2013, August 29) An active processing virtual filesystem for manipulating massive electron microscopy datasets required for connectomics research [Online]. Available:  
<http://people.cs.pitt.edu/~mosse/capstone/2013-fall/Wetzel-CompSci-project-Aug2013.ppt>

[3]D. Stein, (2013, April 20) CS 1687 Capstone Project

[4]M. Szeredi, FUSE – Filesystem in Userspace [Online] Available:  
<http://fuse.sourceforge.net/>