



Deep I/O Performance Analysis of CernVM-FS using Modern Linux Tools

AUGUST 2019

AUTHOR:
Shahnur Isgandarli
EP-SFT

SUPERVISORS:
Jakob Blomer
Gerardo Ganis





PROJECT SPECIFICATION



The CernVM File System provides a scalable, reliable and low-maintenance software distribution service. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on the worldwide-distributed computing infrastructure used to run data processing applications. CernVM-FS is implemented as a POSIX read-only file system in user space (a FUSE module).

The goal of the project is to be able to trace and analyse FUSE calls in CernVM-FS. We would like to connect the dots of user-land and kernel-land performance analysis, answering questions like "how much time did this call spent in the FUSE file system part of the kernel" or "how many times did FUSE kernel callback was executed".

The student should have a good knowledge of C/C++ programming languages, Linux OS and shell scripting as well as tools for software development such as git, cmake, make.

The project should be implemented within period of 9 weeks, from June 17, 2019 to August 16, 2019.





ABSTRACT



This report describes performance analysis of the CernVM-FS FUSE which is a software distribution service used in high-energy physics research. The performance analysis was conducted in both kernel space as well as in userland. One of the main tools used throughout the project implementation is BPF Compiler Collection (BCC). BCC was used for doing performance analysis on the kernel side of the FUSE calls in CernVM-FS. Some new tools were developed for retrieving performance statistics of the FUSE calls based on the guidelines for developing new BCC programs using the python interface. Besides the kernel space, FUSE userland calls were also undergone performance analysis by means of the code instrumentation. Additionally, log2 histogram was merged to the devel branch of the CernVM-FS code repository.





TABLE OF CONTENTS

INTRODUCTION	5
<hr/>	
BACKGROUND	6
FUSE	6
BPF COMPILER COLLETION	6
<hr/>	
PERFORMANCE ENGINEERING OF CERNVM-FS	7
KERNEL-SPACE	7
USERLAND	10
<hr/>	
EXPERIMENTS	12
BENCHMARKING WITH KERNEL-LEVEL CACHING	13
BENCHMARKING WITHOUT KERNEL-LEVEL CACHING	15
<hr/>	
ANALYSIS	17
<hr/>	
CONCLUSION	18
<hr/>	
LIST OF FIGURES	19
<hr/>	
REFERENCES	20





1 Introduction

CernVM-FS is a software distribution service used for the High Energy Physics (HEP) research [1]. As a service, it is quite scalable, reliable and requires low-maintenance. CernVM-FS uses a software interface called “Filesystem in Userspace” (FUSE) that allows non-privileged users to create their own filesystems without touching the kernel-level code [2]. The performance analysis of CernVM-FS was a two-fold problem:

1. Acquisition of performance metrics from the kernel-space.
2. Acquisition of performance metrics from the userland.

The approach for the first problem was to use a set of tools for efficient kernel tracing called “BPF Compiler Collection” (BCC) [3]. The BCC toolset also provides a developer guideline for developing new BCC-based programs using the python interface.

The second problem was solved by means of codebase instrumentation. Timer for measuring the latencies of the CernVM-FS FUSE-related calls were added in the codebase as well as log2 histogram data structure which was need for storing performance metrics and printing out the results of the screen.





2 Background

2.1 FUSE

FUSE (or Filesystem in Userspace) is a software interface for Unix and Unix-like operating systems which allows non-privileged users to create their own file systems without editing the kernel code [2]. FUSE works in the following way (see Figure 1):

1. The request to list the files under “/tmp/fuse” directory gets redirected by kernel through virtual filesystem switch to FUSE interface.
2. FUSE executes the registered handle program which in our case is “./hello” and passes the request to it.
3. The handler program returns a response back to FUSE which is then redirected to the userspace program that originally made the request.

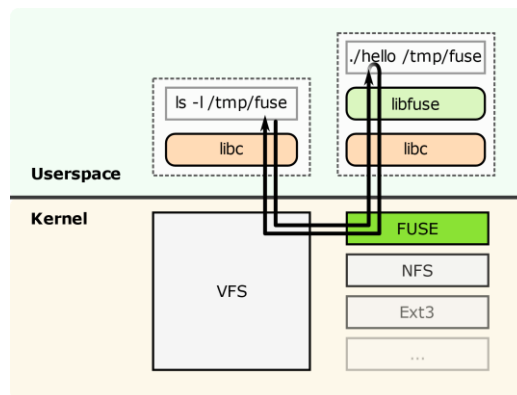


Figure 1. A flow-chart diagram showing how FUSE works¹.

A literature review about the FUSE performance was conducted. The paper [4] describes FUSE technology and its performance overview. The authors found that for many workloads, an optimized FUSE can perform within 5% of the native ext4, however, some workloads are unfriendly to FUSE and can degrade the performance of CPU even if optimized [4].

2.2 BPF Compiler Collection

BPF Compiler Collection (BCC) is set of tools for efficient kernel tracing and performance measurements [3]. This set of tools allows attaching eBPF programs to the kprobe events. Moreover, BCC tools have a python interface for writing own BCC tools. This interface was used for developing the set of scripts used for measuring latencies and counters of FUSE calls throughout the project implementation.

¹ Credits: https://commons.wikimedia.org/wiki/File:FUSE_structure.svg#/media/File:FUSE_structure.svg





3 Performance Engineering of CernVM-FS

3.1 Kernel-space

We started the analysis of performance issues with the kernel-space. The BCC toolset mentioned in the background was very handy in doing the performance measurements and retrieving the statistics of the FUSE calls in the system. All scripts created for the performance engineering of the kernel space are stored in a GitHub repository [5].

Firstly, we needed to collect the latencies of the FUSE calls. Some of the FUSE calls we examined are `dir_open`, `dir_release`, `getattr`, `lookup`, `lookup_name` and `readdir_lat`. The idea behind the script is to create a BPF program with two event handlers: a `kprobe` and a `kretprobe` of FUSE call. Whenever a given kprobe event is fired, it will be caught by the script and the values will be stored internally in the BPF latency histogram. When the user presses CTRL + C, the histogram of the latencies will be printed out on the command line application screen. An example of running `fuse_dir_open_lat.py` and `fuse_lookup_name_lat.py` programs are shown in the figures below (see Figure 2 and Figure 3):

```
[root@fedora latencies]# python fuse_dir_open_lat.py
Tracing... Hit Ctrl-C to end.
^C
```

```

usecs          : count      distribution
  0 -> 1        : 0          |
  2 -> 3        : 0          |
  4 -> 7        : 0          |
  8 -> 15       : 0          |
 16 -> 31       : 0          |
 32 -> 63       : 86         |*****|
 64 -> 127      : 102        |*****|
128 -> 255      : 89         |*****|
256 -> 511      : 47         |*****|
512 -> 1023     : 22         |*****|
1024 -> 2047    : 0          |
2048 -> 4095    : 0          |
4096 -> 8191    : 1          |

```

Figure 2. The output of running `fuse_dir_open_lat.py` program which captures the latencies of FUSE `dir_open` calls. The output is given in a log2 histogram form.

```
[root@fedora latencies]# python fuse_lookup_name_lat.py
Tracing... Hit Ctrl-C to end.
^C
```

```

usecs          : count      distribution
  0 -> 1        : 0          |
  2 -> 3        : 0          |
  4 -> 7        : 0          |
  8 -> 15       : 78         |
 16 -> 31       : 1496       |*****|
 32 -> 63       : 3311       |*****|
 64 -> 127      : 1295       |*****|
128 -> 255      : 46         |
256 -> 511      : 6          |
512 -> 1023     : 2          |
1024 -> 2047    : 1          |

```

Figure 3. The output of running `fuse_lookup_name_lat.py` program which captures the latencies of FUSE `lookup_name` calls. The output is given in a log2 histogram form.





Secondly, we needed to capture the FUSE call counters. The model for capturing the counters is simple: all we need to do is to attach a FUSE call kprobe for a given FUSE call, and when the kprobe is caught by the program, the internal counter will increment the resultant value. At the end of the program execution, the program will print out the number of times a given FUSE call was called.

One of the ways to identify and catch a certain FUSE call is to use regular expression filter for a FUSE call. As an example, let's observe *fuse_dir_open_count.py* file: since we count only *dir_open* events, we need to attach kprobe which *starts and ends with "fuse_dir_open"*. A regular expression for this case is *"^fuse_dir_open\$"*.

Moreover, we needed to retrieve all counters related to the FUSE calls to get the view of all FUSE calls. In order to get all FUSE call counters we used a regular expression *".*fuse.*"* which will filter and show only the FUSE calls. The output of executing *fuse_calls_count.py* is given below: in each line of the output, the address of the FUSE call, FUSE call name and its count are shown (see Figure 4). Moreover, it is possible to export the output into a CSV format via running *fuse_calls_count_csv.py* program which does the same job as *fuse_calls_count.py*, however, instead of printing the output on the terminal, it stores it in a CSV file.

```
[root@fedora fuse_calls_count]# python fuse_calls_count.py
Tracing... Ctrl-C to end.
^C
ADDR                FUNC                COUNT
ffffffffc0a7e201    fuse_init_symlink    2
ffffffffc0a840e1    fuse_conn_get        6
ffffffffc0a7b3f1    fuse_dentry_release  6
ffffffffc0a773b1    fuse_dev_ioctl       6
ffffffffc0a77001    fuse_dev_open        6
ffffffffc0a85241    fuse_dev_alloc       6
ffffffffc0a7d241    fuse_atomic_open     6
ffffffffc0a84781    fuse_show_options    8
ffffffffc0a7bb61    fuse_get_link        18
ffffffffc0a840a1    fuse_inode_eq        40
ffffffffc0a7ba61    fuse_readlink_page   59
ffffffffc0a80141    fuse_do_readpage     71
ffffffffc0a804d1    fuse_readpage        74
ffffffffc0a7b641    fuse_dir_open        326
ffffffffc0a7b621    fuse_dir_release     332
ffffffffc0a7ed21    fuse_file_mmap       349
ffffffffc0a788e1    fuse_get_req         582
ffffffffc0a87421    fuse_readdir         659
ffffffffc0a81cc1    fuse_read_fill       667
ffffffffc0a7e1b1    fuse_init_dir        710
ffffffffc0a7e731    fuse_vma_close       1475
ffffffffc0a83771    fuse_writepages     1531
ffffffffc0a87141    fuse_emit            2130
ffffffffc0a7fa91    fuse_short_read      2650
ffffffffc0a788f1    fuse_get_req_for_background 2994
ffffffffc0a7fc21    fuse_readpages_end   3034
ffffffffc0a80ce1    fuse_open            3169
ffffffffc0a80e01    fuse_release         3252
ffffffffc0a7e1a1    fuse_init_common     3439
ffffffffc0a84021    fuse_init_file_inode 3439
ffffffffc0a7ceal    fuse_invalidate_atime 3447
ffffffffc0a7fd91    fuse_send_readpages.isra.0 3509
ffffffffc0a80d61    fuse_release_common  3557
ffffffffc0a80af1    fuse_finish_open     3735
ffffffffc0a7e9f1    fuse_send_open.isra.0 3757
ffffffffc0a80be1    fuse_open_common     3775
```

Figure 4. The output of running *fuse_calls_count.py* program which captures the FUSE call counters.





Another script called *show_diff.py* was developed in order to show the differences between the experiment results stored in a CSV files. The output of running the *show_diff.py* program is given below (see Figure 5). The colors mean the differences between the metrics in two different experiments:

- if the difference is less or equal than 100, then the color is green;
- if it is less or equal than 500, then the color is blue;
- otherwise, the color is yellow.

```
[root@fedora fuse_calls_count]# python show_diff.py file1 file2
fuse_iget | 3792 | 4824
fuse_writepages | 1418 | 1405
fuse_readlink_page | 102 | 102
fuse_conn_init | 1 | 0
fuse_conn_put | 0 | 2
fuse_copy_do | 1361683 | 1373522
fuse_read_fill | 740 | 740
fuse_emit | 5741 | 5747
fuse_show_options | 140 | 165
fuse_queue_forget | 3710 | 5274
fuse_copy_args | 623827 | 624753
fuse_dev_do_write | 314561 | 314786
fuse_lookup | 6693 | 7713
fuse_invalidate_atime | 5950 | 6459
fuse_request_send_background | 9400 | 9908
fuse_request_queue_background | 9400 | 9908
fuse_dev_open | 5 | 2
fuse_conn_get | 4 | 2
fuse_find_writeback | 55775 | 60237
fuse_copy_fill | 988405 | 994049
fuse_invalidate_attr | 260 | 4
fuse_readpages | 5133 | 5635
fuse_lookup_name | 6693 | 7713
fuse_release_end | 4265 | 4266
fuse_do_open | 4266 | 4266
fuse_request_alloc | 4267 | 4266
fuse_file_read_iter | 6413 | 6413
fuse_file_mmap | 438 | 438
fuse_file_alloc | 4266 | 4266
fuse_request_free | 314302 | 314785
fuse_file_get | 5133 | 5642
__fuse_request_send | 304902 | 304876
fuse_put_request | 933108 | 934437
fuse_fill_super | 1 | 0
fuse_perm_getattr | 132795 | 132774
fuse_get_root_inode | 1 | 0
fuse_init_symlink | 24 | 33
fuse_dentry_release | 4957 | 4825
fuse_init_file_inode | 2976 | 3865
fuse_valid_type | 1 | 0
fuse_dev_free | 0 | 2
fuse_send_readpages.isra.0 | 5133 | 5642
```

Figure 5. The output of running *show_diff.py* command with two arguments which are outputs of two different experiments in a CSV format.





3.2 Userland

The performance engineering of FUSE userland calls in CernVM-FS required a different set of techniques called *code instrumentation*. The goal was to measure the latencies the following CernVM-FS FUSE calls:

- `cvmfs_lookup`
- `cvmfs_forget`
- `cvmfs_getattr`
- `cvmfs_readlink`
- `cvmfs_opendir`
- `cvmfs_releasedir`
- `cvmfs_readdir`
- `cvmfs_open`
- `cvmfs_read`
- `cvmfs_release`

The main problem was a lack of data structure to store the latency calls. A log2 histogram data structure was developed for storing the latencies of FUSE userland calls in CernVM-FS.

Another issue was the way to measure the latency of calls. *HighPrecisionTimer* class was used for measuring the latency in order to retrieve the accurate results. You can see the *HighPrecisionTimer* class below (see Figure 6):

```
class HighPrecisionTimer : SingleCopy {
public:
    explicit HighPrecisionTimer(Log2Histogram *recorder)
        : timestamp_start_(platform_monotonic_time_ns())
        , recorder_(recorder)
    { }

    ~HighPrecisionTimer() {
        recorder_>Add(platform_monotonic_time_ns() - timestamp_start_);
    }

private:
    uint64_t timestamp_start_;
    Log2Histogram *recorder_;
};
```

Figure 6. *HighPrecisionTimer* class used for latency measurements.

A *HighPrecisionTimer* constructor accepts a pointer to a *Log2Histogram* data structure which stores all latency measurements. Upon initialization, the timer initializes the *timestamp_start_* variable to the current time in nanoseconds via usage of *platform_monotonic_time_ns* function which returns the time in nanoseconds. Upon destruction, the timer updates the histogram by adding the difference of current time and start time.

To measure the latencies of CernVM-FS functions, we added instances of *HighPrecisionTimer* as a first line in all functions related to CernVM-FS logic that we examine. Since the timer accepts an argument of *Log2Histogram* type, we pass the pointer to the associated *Log2Histogram* instance, and upon leave from the function, the timer will store a record in the *Log2Histogram* instance passed as an argument to the constructor. The figure below shows the implementation of latency measurement of *cvmfs_lookup* function (see Figure 7):





```
static void cvmfs_lookup(/* cvmfs_lookup function args */) {
    HighPrecisionTimer(file_system_>hist_fs_lookup());

    /* cvmfs_lookup function logic */
}
```

Figure 7. Skeleton of the `cvmfs_lookup` function with a usage of `HighPrecisionTimer`.

The last parts of latency measurements of CernVM-FS functions in userland were measuring the overhead of the `HighPrecisionTimer` and showing the results on the screen.

Timer overhead measurement was achieved via running the benchmark with a modified implementation of the `HighPrecisionTimer`: there were no need to update histogram, instead the latency was just calculated and not recorded at all. In the figure below, the measurements of timer overhead are shown (see Figure 8).

	nsec	count	distribution
0 ->	1 :	0	
2 ->	3 :	0	
4 ->	7 :	0	
8 ->	15 :	0	
16 ->	31 :	5370	****
32 ->	63 :	15081	*****
64 ->	127 :	23070	*****
128 ->	255 :	1587	*
256 ->	511 :	5017	***
512 ->	1023 :	1	
1024 ->	2047 :	0	
2048 ->	4095 :	0	
4096 ->	8191 :	1	
8192 ->	16383 :	31	
16384 ->	32767 :	2	
32768 ->	65535 :	0	
65536 ->	131071 :	0	
131072 ->	262143 :	0	
262144 ->	524287 :	0	
524288 ->	1048575 :	0	
	overflow :	0	
	total :	50160	

Figure 8. Timer overhead while measuring the latencies of CernVM-FS function calls.

As can be seen from latency distribution shown in the Figure 8, most of the calls in CernVM-FS spent between 64 nanoseconds to 127 nanoseconds on latency measurement.

The command `cvmfs_talk internal affairs` is used to view the latencies of examined functions. It prints the internal status information and performance counters. It can be helpful for performance engineering [6].





4 Experiments

The next stage of the project was implementing experiments and retrieving the results. Two main experiment types were the following:

- Running the benchmark *with the kernel level-caching*.
- Running the benchmark *without the kernel-level caching*.





4.1 Benchmarking with Kernel-level Caching

Below, you can see the latencies of CernVM-FS's FUSE related function calls we got by benchmarking *with the kernel-level caching* (see Figure 9):

Lookup				Readlink			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	0		2048 ->	4095	2	
4096 ->	8191	41		4096 ->	8191	30	*****
8192 ->	16383	473	**	8192 ->	16383	23	*****
16384 ->	32767	2023	*****	16384 ->	32767	28	*****
32768 ->	65535	3176	*****	32768 ->	65535	17	*****
65536 ->	131071	1872	*****	65536 ->	131071	2	
131072 ->	262143	77		131072 ->	262143	0	
262144 ->	524287	5		262144 ->	524287	0	
524288 ->	1048575	1		524288 ->	1048575	0	
1048576 ->	2097151	2		1048576 ->	2097151	0	
2097152 ->	4194303	0		2097152 ->	4194303	0	
4194304 ->	8388607	0		4194304 ->	8388607	0	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	1		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	0	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow ->		0		overflow ->		0	
total :		7671		total :		102	

Forget				Opendir			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	0		2048 ->	4095	0	
4096 ->	8191	0		4096 ->	8191	0	
8192 ->	16383	9		8192 ->	16383	0	
16384 ->	32767	0		16384 ->	32767	21	**
32768 ->	65535	0		32768 ->	65535	67	*****
65536 ->	131071	0		65536 ->	131071	104	*****
131072 ->	262143	0		131072 ->	262143	94	*****
262144 ->	524287	0		262144 ->	524287	50	*****
524288 ->	1048575	0		524288 ->	1048575	18	*****
1048576 ->	2097151	0		1048576 ->	2097151	6	*
2097152 ->	4194303	0		2097152 ->	4194303	1	
4194304 ->	8388607	0		4194304 ->	8388607	3	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	0		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	1	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow ->		0		overflow ->		0	
total :		0		total :		365	

Getattr				Releasedir			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	68	*****
2048 ->	4095	0		2048 ->	4095	169	*****
4096 ->	8191	0		4096 ->	8191	101	*****
8192 ->	16383	0		8192 ->	16383	20	**
16384 ->	32767	0		16384 ->	32767	7	
32768 ->	65535	0		32768 ->	65535	0	
65536 ->	131071	1	*****	65536 ->	131071	0	
131072 ->	262143	1	*****	131072 ->	262143	0	
262144 ->	524287	0		262144 ->	524287	0	
524288 ->	1048575	0		524288 ->	1048575	0	
1048576 ->	2097151	0		1048576 ->	2097151	0	
2097152 ->	4194303	0		2097152 ->	4194303	0	
4194304 ->	8388607	0		4194304 ->	8388607	0	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	0		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	0	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow ->		0		overflow ->		0	
total :		2		total :		365	





Readdir				Read			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	307	*****	2048 ->	4095	2	
4096 ->	8191	266	*****	4096 ->	8191	123	
8192 ->	16383	94	****	8192 ->	16383	958	*****
16384 ->	32767	69	***	16384 ->	32767	1582	*****
32768 ->	65535	2		32768 ->	65535	1610	*****
65536 ->	131071	0		65536 ->	131071	348	**
131072 ->	262143	0		131072 ->	262143	28	
262144 ->	524287	0		262144 ->	524287	0	
524288 ->	1048575	0		524288 ->	1048575	1	
1048576 ->	2097151	0		1048576 ->	2097151	0	
2097152 ->	4194303	0		2097152 ->	4194303	0	
4194304 ->	8388607	0		4194304 ->	8388607	0	
8388608 ->	16777215	0		8388608 ->	16777215	1	
16777216 ->	33554431	0		16777216 ->	33554431	12	
33554432 ->	67108863	0		33554432 ->	67108863	18	
67108864 ->	134217727	0		67108864 ->	134217727	2	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow :	0	0		overflow :	0	0	
total :		738		total :		4685	

Open				Release			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	0		2048 ->	4095	390	***
4096 ->	8191	0		4096 ->	8191	1643	*****
8192 ->	16383	27		8192 ->	16383	1434	*****
16384 ->	32767	114	*	16384 ->	32767	424	****
32768 ->	65535	250	**	32768 ->	65535	8	
65536 ->	131071	21		65536 ->	131071	0	
131072 ->	262143	2		131072 ->	262143	0	
262144 ->	524287	0		262144 ->	524287	0	
524288 ->	1048575	873	*****	524288 ->	1048575	0	
1048576 ->	2097151	1236	*****	1048576 ->	2097151	0	
2097152 ->	4194303	114	*	2097152 ->	4194303	0	
4194304 ->	8388607	47		4194304 ->	8388607	0	
8388608 ->	16777215	20		8388608 ->	16777215	0	
16777216 ->	33554431	752	*****	16777216 ->	33554431	0	
33554432 ->	67108863	383	***	33554432 ->	67108863	0	
67108864 ->	134217727	33		67108864 ->	134217727	0	
134217728 ->	268435455	15		134217728 ->	268435455	0	
268435456 ->	536870911	9		268435456 ->	536870911	0	
536870912 ->	1073741823	3		536870912 ->	1073741823	0	
overflow :	0	0		overflow :	0	0	
total :		3899		total :		3899	

Figure 9. Latencies of CernVM-FS's FUSE related function calls with kernel-level caching turned on.





4.2 Benchmarking without Kernel-level Caching

Below, you can see the latencies of CernVM-FS's FUSE related function calls we got by benchmarking *without the kernel-level caching*:

Lookup				Readlink			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	0		2048 ->	4095	9	***
4096 ->	8191	49504	*****	4096 ->	8191	30	*****
8192 ->	16383	59227	*****	8192 ->	16383	26	*****
16384 ->	32767	22538	*****	16384 ->	32767	13	*****
32768 ->	65535	5953	*	32768 ->	65535	17	*****
65536 ->	131071	976		65536 ->	131071	7	**
131072 ->	262143	34		131072 ->	262143	0	
262144 ->	524287	1		262144 ->	524287	0	
524288 ->	1048575	0		524288 ->	1048575	0	
1048576 ->	2097151	1		1048576 ->	2097151	0	
2097152 ->	4194303	0		2097152 ->	4194303	0	
4194304 ->	8388607	0		4194304 ->	8388607	0	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	1		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	0	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow		0		overflow		0	
total		138235		total		102	

Forget				Opendir			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	0		2048 ->	4095	0	
4096 ->	8191	0		4096 ->	8191	0	
8192 ->	16383	0		8192 ->	16383	0	
16384 ->	32767	0		16384 ->	32767	4	
32768 ->	65535	0		32768 ->	65535	67	*****
65536 ->	131071	0		65536 ->	131071	95	*****
131072 ->	262143	0		131072 ->	262143	108	*****
262144 ->	524287	0		262144 ->	524287	51	*****
524288 ->	1048575	0		524288 ->	1048575	31	***
1048576 ->	2097151	0		1048576 ->	2097151	4	
2097152 ->	4194303	0		2097152 ->	4194303	3	
4194304 ->	8388607	0		4194304 ->	8388607	1	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	0		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	1	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow		0		overflow		0	
total		0		total		365	

Getattr				Releasedir			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	12	*
2048 ->	4095	33267	*****	2048 ->	4095	200	*****
4096 ->	8191	75062	*****	4096 ->	8191	136	*****
8192 ->	16383	34427	*****	8192 ->	16383	12	*
16384 ->	32767	12750	***	16384 ->	32767	5	
32768 ->	65535	683		32768 ->	65535	0	
65536 ->	131071	33		65536 ->	131071	0	
131072 ->	262143	12		131072 ->	262143	0	
262144 ->	524287	1		262144 ->	524287	0	
524288 ->	1048575	1		524288 ->	1048575	0	
1048576 ->	2097151	0		1048576 ->	2097151	0	
2097152 ->	4194303	0		2097152 ->	4194303	0	
4194304 ->	8388607	1		4194304 ->	8388607	0	
8388608 ->	16777215	0		8388608 ->	16777215	0	
16777216 ->	33554431	0		16777216 ->	33554431	0	
33554432 ->	67108863	0		33554432 ->	67108863	0	
67108864 ->	134217727	0		67108864 ->	134217727	0	
134217728 ->	268435455	0		134217728 ->	268435455	0	
268435456 ->	536870911	0		268435456 ->	536870911	0	
536870912 ->	1073741823	0		536870912 ->	1073741823	0	
overflow		0		overflow		0	
total		156237		total		365	





Readdir				Read			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	0	
2048 ->	4095	211	*****	2048 ->	4095	2	
4096 ->	8191	420	*****	4096 ->	8191	96	
8192 ->	16383	75	*****	8192 ->	16383	884	*****
16384 ->	32767	31	*****	16384 ->	32767	1579	*****
32768 ->	65535	1	*****	32768 ->	65535	1641	*****
65536 ->	131071	0	*****	65536 ->	131071	428	*****
131072 ->	262143	0	*****	131072 ->	262143	21	*****
262144 ->	524287	0	*****	262144 ->	524287	1	*****
524288 ->	1048575	0	*****	524288 ->	1048575	0	*****
1048576 ->	2097151	0	*****	1048576 ->	2097151	0	*****
2097152 ->	4194303	0	*****	2097152 ->	4194303	1	*****
4194304 ->	8388607	0	*****	4194304 ->	8388607	0	*****
8388608 ->	16777215	0	*****	8388608 ->	16777215	0	*****
16777216 ->	33554431	0	*****	16777216 ->	33554431	11	*****
33554432 ->	67108863	0	*****	33554432 ->	67108863	17	*****
67108864 ->	134217727	0	*****	67108864 ->	134217727	3	*****
134217728 ->	268435455	0	*****	134217728 ->	268435455	1	*****
268435456 ->	536870911	0	*****	268435456 ->	536870911	0	*****
536870912 ->	1073741823	0	*****	536870912 ->	1073741823	0	*****
overflow :		0		overflow :		0	
total :		738		total :		4685	

Open				Release			
	nsec	count	distribution		nsec	count	distribution
0 ->	1	0		0 ->	1	0	
2 ->	3	0		2 ->	3	0	
4 ->	7	0		4 ->	7	0	
8 ->	15	0		8 ->	15	0	
16 ->	31	0		16 ->	31	0	
32 ->	63	0		32 ->	63	0	
64 ->	127	0		64 ->	127	0	
128 ->	255	0		128 ->	255	0	
256 ->	511	0		256 ->	511	0	
512 ->	1023	0		512 ->	1023	0	
1024 ->	2047	0		1024 ->	2047	1	
2048 ->	4095	0		2048 ->	4095	406	*****
4096 ->	8191	0		4096 ->	8191	1537	*****
8192 ->	16383	27	*****	8192 ->	16383	1514	*****
16384 ->	32767	155	*****	16384 ->	32767	425	*****
32768 ->	65535	226	*****	32768 ->	65535	15	*****
65536 ->	131071	5	*****	65536 ->	131071	0	*****
131072 ->	262143	0	*****	131072 ->	262143	1	*****
262144 ->	524287	1	*****	262144 ->	524287	0	*****
524288 ->	1048575	494	*****	524288 ->	1048575	0	*****
1048576 ->	2097151	1718	*****	1048576 ->	2097151	0	*****
2097152 ->	4194303	108	*****	2097152 ->	4194303	0	*****
4194304 ->	8388607	76	*****	4194304 ->	8388607	0	*****
8388608 ->	16777215	23	*****	8388608 ->	16777215	0	*****
16777216 ->	33554431	682	*****	16777216 ->	33554431	0	*****
33554432 ->	67108863	333	*****	33554432 ->	67108863	0	*****
67108864 ->	134217727	24	*****	67108864 ->	134217727	0	*****
134217728 ->	268435455	17	*****	134217728 ->	268435455	0	*****
268435456 ->	536870911	8	*****	268435456 ->	536870911	0	*****
536870912 ->	1073741823	1	*****	536870912 ->	1073741823	0	*****
overflow :		1		overflow :		0	
total :		3899		total :		3899	

Figure 10. Latencies of CernVM-FS's FUSE related function calls with kernel-level caching turned off.





5 Analysis

The focus of analysis was given to *lookup calls* in CernVM-FS. The results on the left are the latencies of lookup calls when the kernel-level caching is turned on; the results on the right are the latencies of the lookup calls when the kernel-level caching is turned off (see Figure 11).

As can be seen from the figure below (see Figure 11), the software spends more time on executing the lookup calls in the system with a kernel-level caching enabled in the system: the majority of the calls take around 65µs-131µs. The number of calls made are low due to having caching mechanism.

Unlike kernel-level caching, the plain system setup without kernel-level caching *accelerates* most of lookup calls in CernVM-FS from latencies of 65µs-131µs to 32µs-65µs. However, due to having the caching mechanism in a disabled state, the number of calls made in the system have increased drastically: as can be seen from the figure, the number of calls increased by ≈18 times.

Lookup	nsec	count	distribution
0 ->	1	0	
2 ->	3	0	
4 ->	7	0	
8 ->	15	0	
16 ->	31	0	
32 ->	63	0	
64 ->	127	0	
128 ->	255	0	
256 ->	511	0	
512 ->	1023	0	
1024 ->	2047	0	
2048 ->	4095	0	
4096 ->	8191	0	
8192 ->	16383	2142	
16384 ->	32767	12539	***
32768 ->	65535	106964	*****
65536 ->	131071	15829	****
131072 ->	262143	532	
262144 ->	524287	124	
524288 ->	1048575	5	
1048576 ->	2097151	4	
2097152 ->	4194303	10	
4194304 ->	8388607	3	
8388608 ->	16777215	1	
16777216 ->	33554431	0	
33554432 ->	67108863	0	
67108864 ->	134217727	0	
134217728 ->	268435455	0	
268435456 ->	536870911	0	
536870912 ->	1073741823	0	
overflow		0	
total		138235	

Without kernel-level caching

Lookup	nsec	count	distribution
0 ->	1	0	
2 ->	3	0	
4 ->	7	0	
8 ->	15	0	
16 ->	31	0	
32 ->	63	0	
64 ->	127	0	
128 ->	255	0	
256 ->	511	0	
512 ->	1023	0	
1024 ->	2047	0	
2048 ->	4095	0	
4096 ->	8191	0	
8192 ->	16383	1	
16384 ->	32767	34	
32768 ->	65535	1226	*****
65536 ->	131071	5148	*****
131072 ->	262143	1253	*****
262144 ->	524287	0	
524288 ->	1048575	0	
1048576 ->	2097151	0	
2097152 ->	4194303	0	
4194304 ->	8388607	1	
8388608 ->	16777215	0	
16777216 ->	33554431	0	
33554432 ->	67108863	0	
67108864 ->	134217727	0	
134217728 ->	268435455	0	
268435456 ->	536870911	0	
536870912 ->	1073741823	0	
overflow		0	
total		7663	

With kernel-level caching

Figure 11. The comparison of lookup calls' latencies in two different configurations: with kernel-level caching and without kernel-level caching.





6 Conclusion

In conclusion, this report summarizes the implementation of the project called “Deep I/O Performance Analysis of CernVM-FS using the Modern Linux Tools”. After implementation of the project, we have the got the following deliverables:

- A powerful set of tools to look in user and kernel spaces of FUSE calls.
- A toolset that enables fine-grained performance engineering of CernVM-FS client.
- log2 histogram data structure for latency measurements which is merged in CernVM-FS into the CernVM-FS devel branch.





7 List of figures

Figure 1. A flow-chart diagram showing how FUSE works.....6

Figure 2. The output of running fuse_dir_open_lat.py program which captures the latencies of FUSE dir_open calls. The output is given in a log2 histogram form.7

Figure 3. The output of running fuse_lookup_name_lat.py program which captures the latencies of FUSE lookup_name calls. The output is given in a log2 histogram form.7

Figure 4. The output of running fuse_calls_count.py program which captures the FUSE call counters.8

Figure 5. The output of running show_diff.py command with two arguments which are outputs of two different experiments in a CSV format.....9

Figure 6. HighPrecisionTimer class used for latency measurements..... 10

Figure 7. Skeleton of the cvmfs_lookup function with a usage of HighPrecisionTimer. 11

Figure 8. Timer overhead while measuring the latencies of CernVM-FS function calls..... 11

Figure 9. Latencies of CernVM-FS’s FUSE related function calls with kernel-level caching turned on. 14

Figure 10. Latencies of CernVM-FS’s FUSE related function calls with kernel-level caching turned off. 16

Figure 11. The comparison of lookup calls’ latencies in two different configurations: with kernel-level caching and without kernel-level caching. 17





8 References

1. *CernVM File System*. Available from: <https://cernvm.cern.ch/portal/filesystem>.
2. Wikipedia. *Filesystem in Userspace* Available from: https://en.wikipedia.org/wiki/Filesystem_in_Userspace.
3. Iovisor. *iovisor/bcc*. Available from: <https://github.com/iovisor/bcc>.
4. Vangoor, B.K.R., V. Tarasov, and E. Zadok. *To FUSE or Not to FUSE: Performance of User-Space File Systems*. in *15th {USENIX} Conference on File and Storage Technologies ({FAST} 17)*. 2017. Santa Clara, CA: {USENIX} Association.
5. Isgandarli, S. *sisgandarli/cvmfs_fuse_perf*. Available from: https://github.com/sisgandarli/cvmfs_fuse_perf.
6. *Client Configuration — CernVM-FS 2.7.0 documentation*. Available from: <https://cvmfs.readthedocs.io/en/stable/cpt-configure.html>.

