rfs – Remote File System
Softwarepraktikum für Fortgeschrittene

Parallele und Verteilte Systeme
Institut für Informatik
Ruprecht-Karls-Universität Heidelberg

Michael Kuhn
Betreuer: Julian Kunkel

2009-07-09

Contents

1. Introduction

1.1. Goals

The goal of this practical is to implement a network file system daemon and a global network file system on top of it. For the global network file system the FUSE[1] framework is used. FUSE file systems themselves run in user space and use the special device `/dev/fuse` to communicate with the kernel part of FUSE.

1.2. Requirements

The network file system daemon needs to support separate control and data channels to allow encrypted authentication over the control channel and unencrypted file transfer over the data channel.

[1]Filesystem in Userspace – `http://fuse.sourceforge.net/`

2. Overview

The Remote File System consists of three major components.

2.1. Remote File System Daemon

The Remote File System Daemon provides access to the local file system over the network. It features separate control and data channels as well as multiple concurrent sessions.

2.2. Remote File System Library

The Remote File System Library provides an easy-to-use API to communicate with the Remote File System Daemon.

2.3. Global Remote File System

The Global Remote File System provides a global namespace.

3. Remote File System Daemon

3.1. Motivation

A separate protocol was designed, because existing protocols did not meet the requirements. The SSH and FTP protocols were considered.
Problems with the SSH protocol include:

- SSH does not support unencrypted data channels.
- Data encryption makes transfers too slow.
  It is also not possible to deactivate the encryption.

Problems with the FTP protocol include:

- It is only possible to write a complete file or append data to it.
- File listings are hard to parse, because their format is not well-defined.
  The FTP RFC states "Since the information on a file may vary widely from system to system, this information may be hard to use automatically in a program, but may be quite useful to a human user."

3.2. Features

The Remote File System Daemon is fully multi-threaded, with each connection handled by its own thread. Therefore, long-running operations do not block any other connections.
It includes a `chroot` feature to restrict users to a specified sub-tree of the file system and can execute file system commands with different user and group permissions.
Replication can be used to duplicate the file system on another machine. Each Remote File System Daemon can be the *master* of several *slaves*.
Should a slave go offline all operations are logged by the master so they can be replayed later. Each failed operation is kept in memory and written into a log file. Should the master itself crash or be shut down all operations can be read back from the log file and then be replayed.

3.3. Configuration

The Remote File System Daemon expects its password and master password in `${XDG_CONFIG_HOME}`[1]`/rfs/rfsd/password` and `${XDG_CONFIG_HOME}/rfs/rfsd/master_password` respectively.

[1]If unset, `${XDG_CONFIG_HOME}` defaults to `${HOME}/.config`.

3.4. Protocol

The client sends one command and then receives a response from the daemon. The daemon always responds with either `OK` or `FAIL`.

```
> COMMAND arguments
< OK [arguments]
```

```
> COMMAND arguments
< FAIL [error]
```

3.5. Commands

```
> LOGIN password
< OK port token
```

Logs in using *password*. Returns the data channel's port and the authentication token.

```
> QUIT
```

Closes the connection.

```
> MASTER password
< OK
```

Logs in as master using *password*.

```
> READ file
< OK
```

Open *file* for reading. The area to read and the data is sent over the data channel.

```
> WRITE file
< OK
```

Opens *file* for writing. The area to write and the data is sent over the data channel.

```
> READDIR directory
< OK
```

Returns the names of all entries in *directory*. The names are sent over the data channel.

```
> STAT file
< OK
```

Returns metadata of *file*. The metadata is sent over the data channel.

```
> MKDIR directory
< OK
```

Creates *directory*.

```
> RMDIR directory
< OK
```

Removes *directory*.

```
> CREATE file
< OK
```

Creates *file*.

```
> UNLINK file
< OK
```

Removes *file*.

```
> TRUNCATE size file
< OK
```

Truncates *file* to size *size*.

```
> ACCESS mode file
< OK
```

Checks *file* for accessibility as defined by *mode*.

```
> CHMOD mode file
< OK
```

Changes the mode of *file* to *mode*.

```
> CHOWN uid gid file
< OK
```

Changes the owner of *file* to user ID *uid* and group ID *gid*.

3.6. Example

A typical session looks like the following.

```
> LOGIN secret
< OK 6666 3a4d54e553b26a6eb720a4be6033482b
> READDIR /
< OK
> STAT /tmp
< OK
> READDIR /tmp
< OK
```

```
> CREATE /tmp/rfs
< OK
> WRITE /tmp/rfs
< OK
> UNLINK /tmp/rfs
< OK
> QUIT
```

4. Remote File System Library

4.1. Application Programming Interface

4.1.1. Functions

```
rfs* rfs_new (void);
```

This function creates a new `rfs` object.

```
void rfs_free (rfs*);
```

This function frees an existing `rfs` object.

```
gboolean rfs_connect (rfs*, const gchar* host, guint port);
```

This function connects to the specified host on the specified port.

```
void rfs_disconnect (rfs*);
```

This function disconnects from the connected host.

```
rfsStatus rfs_login (rfs*, const gchar* password, GError**);
```

This function logs in with the specified password.

```
rfsStatus rfs_master (rfs*, const gchar* password, GError**);
```

This function logs in as master with the specified password and is usually only used internally.

```
rfsStatus rfs_read (rfs*, const gchar* path, GError**);
rfsStatus rfs_read_do (rfs*, gchar* buffer, gsize size, goffset offset,
                       gsize* bytes_read, GError**);
rfsStatus rfs_read_end (rfs*, GError**);
```

These functions read data from the specified file. `rfs_read` opens the file for reading, `rfs_read_do` reads data from `offset` into `buffer` of size `size`. `rfs_read_end` closes the file.

```
rfsStatus rfs_write (rfs*, const gchar* path, GError**);
rfsStatus rfs_write_do (rfs*, const gchar* buffer, gsize size, goffset offset,
                        gsize* bytes_written, GError**);
rfsStatus rfs_write_end (rfs*, GError**);
```

These functions write data to the specified file. `rfs_write` opens the file for writing, `rfs_write_do` writes data from `buffer` of size `size` at `offset`. `rfs_write_end` closes the file.

```
rfsStatus rfs_readdir (rfs*, const gchar* path, GError**);
rfsStatus rfs_readdir_do (rfs*, gchar* filename, GError**);
```

These functions read the specified directory. `rfs_readdir` opens the directory for reading, `rfs_readdir_do` reads a filename.

```
rfsStatus rfs_stat (rfs*, const gchar* path, rfsStat* stat, GError**);
```

This function returns information about the specified file.

```
rfsStatus rfs_mkdir (rfs*, const gchar* path, GError**);
```

This function creates the specified directory.

```
rfsStatus rfs_rmdir (rfs*, const gchar* path, GError**);
```

This function removes the specified directory.

```
rfsStatus rfs_create (rfs*, const gchar* path, GError**);
```

This function creates the specified file.

```
rfsStatus rfs_unlink (rfs*, const gchar* path, GError**);
```

This function removes the specified file.

```
rfsStatus rfs_truncate (rfs*, const gchar* path, goffset size, GError**);
```

This function truncates the specified file to `size`.

```
rfsStatus rfs_access (rfs*, const gchar* path, guint flags, GError**);
```

This function checks access permissions for the specified file.

```
rfsStatus rfs_chmod (rfs*, const gchar* path, gint mode, GError**);
```

This function changes the mode of the specified file.

```
rfsStatus rfs_chown (rfs*, const gchar* path, gint uid, gint gid, GError**);
```

This function changes the user and group of the specified file.

```
rfsStatus rfs_quit (rfs*, GError**);
```

This function logs out.

```
void rfs_id (rfs*, gint uid, gint gid);
```

This function sets the user and group for all following commands and is usually only used internally.

4.1.2. Structures

```
#define RFS_ERROR rfs_error_quark()
```

This specifies the error domain as used in the `GError` objects returned by the functions above.

```
typedef struct rfs rfs;
```

This specifies an `rfs` object.

```
typedef struct
{
    goffset size;
    gint64 atime;
    gint64 mtime;
    gint64 ctime;
    guint64 nlink;
    guint32 mode;
    guint32 uid;
    guint32 gid;
}
rfsStat;
```

This specifies the structure passed to `rfs_stat`.

```
enum
{
    RFS_ACCESS_F = (1 << 0),
    RFS_ACCESS_R = (1 << 1),
    RFS_ACCESS_W = (1 << 2),
    RFS_ACCESS_X = (1 << 3)
};
```

This specifies the flags passed to `rfs_access`.

```
typedef enum
{
    RFS_STATUS_ERROR,
    RFS_STATUS_OK,
    RFS_STATUS_EOF
}
rfsStatus;
```

This specifies the statuses returned by the functions above.

```
typedef enum
{
    RFS_ERROR_DISCONNECTED,
    RFS_ERROR_FAILED
}
rfsError;
```

This specifies the error codes as used in the `GError` objects returned by the functions above.

4.2. Remote File System Client

This library is used by the Remote File System Client – `rfsc` – to test the Remote File System in an automated way and provide some performance results.

5. Global Remote File System

5.1. Concept

The Global Remote File System can be used to merge multiple file systems provided by Remote File Systems Daemons into one global namespace.

5.2. Features

The Global Remote File System supports High Availability and continues to work if servers go offline. It then simply uses the remaining servers, providing a partial view of the global file system.
Read and write operations are implemented in such a way that subsequent reads from or writes to the same file are handled with very little overhead and are therefore as fast as possible.

5.3. Configuration

The Global Remote File System expects the password for a Remote File System Daemon running on host `host` in `${XDG_CONFIG_HOME}/rfs/grfs/host`.

5.4. Example

Let `host1` have the file `/tmp/rfs1`, `host2` the file `/tmp/rfs2` and `host3` the file `/tmp/rfs3`.

```
$ grfs host1:6666 host2:6666 host3:6666 /grfs
$ ls /grfs/tmp
rfs1 rfs2 rfs3
```

6. Evaluation

6.1. ViroQuant

Evaluation was done with 12 machines. Each machine runs several Xen instances:

- `vqstorNN`: Hosts the Remote File System Daemons
- `vqcompNN`: Hosts the Remote File System Clients

Each Xen instance has access to 8 Intel Xeon CPUs with 2.33 GHz and 4 GByte of RAM. The Remote File System Daemons used an XFS file system of size 5.5 TBytes.
All number are per-node, except for aggregated numbers.

6.1.1. 12 Daemons & 12 Clients

Without Replication

One Remote File System Daemon on each `vqstor`. One Remote File System Client on each `vqcomp`.

Operation	Ops/s
`create`	4,281
`stat`	5,053
`access`	5,434
`chmod`	4,904
`chown`	5,249
`unlink`	4,498

Operation	MB/s	Aggregated MB/s
`write`	81	968
`read`	88	1067

With Replication

Two Remote File System Daemons on each `vqstor`. One Remote File System Client on each `vqcomp`.

Operation	Ops/s
`create`	1,930
`stat`	5,015
`access`	5,446
`chmod`	2,295
`chown`	2,325
`unlink`	1,992

Operation	MB/s	Aggregated MB/s
`write`	36	437
`read`	92	1106

6.1.2. 6 Daemons & 6 Clients

With Replication

Two Remote File System Daemons on each of `vqstor01` to `vqstor06`. One Remote File System Client on each of `vqcomp07` to `vqcomp12`.

Operation	Ops/s
`create`	1,851
`stat`	4,308
`access`	4,823
`chmod`	2,294
`chown`	2,310
`unlink`	2,012

Operation	MB/s	Aggregated MB/s
`write`	36	216
`read`	107	640

6.1.3. Global Remote File System

6 Daemons & 6 Clients

One Remote File System Daemon on each of `vqstor01` to `vqstor06`. One Global Remote File System on `vqstor07`, serving six clients.

Normal I/O

Operation	Ops/s
`create`	364
`unlink`	396

Operation	MB/s	Aggregated MB/s
`write`	8	48
`read`	19	114

Direct I/O

Operation	Ops/s
`create`	347
`unlink`	430

Operation	MB/s	Aggregated MB/s
`write`	19	114
`read`	18	108

1 Daemon & 1 Client

One Remote File System Daemon on `vqstor01`. One Global Remote File System on `vqstor07`, serving one client.

Normal I/O

Operation	Ops/s
`create`	1,326
`unlink`	1,834

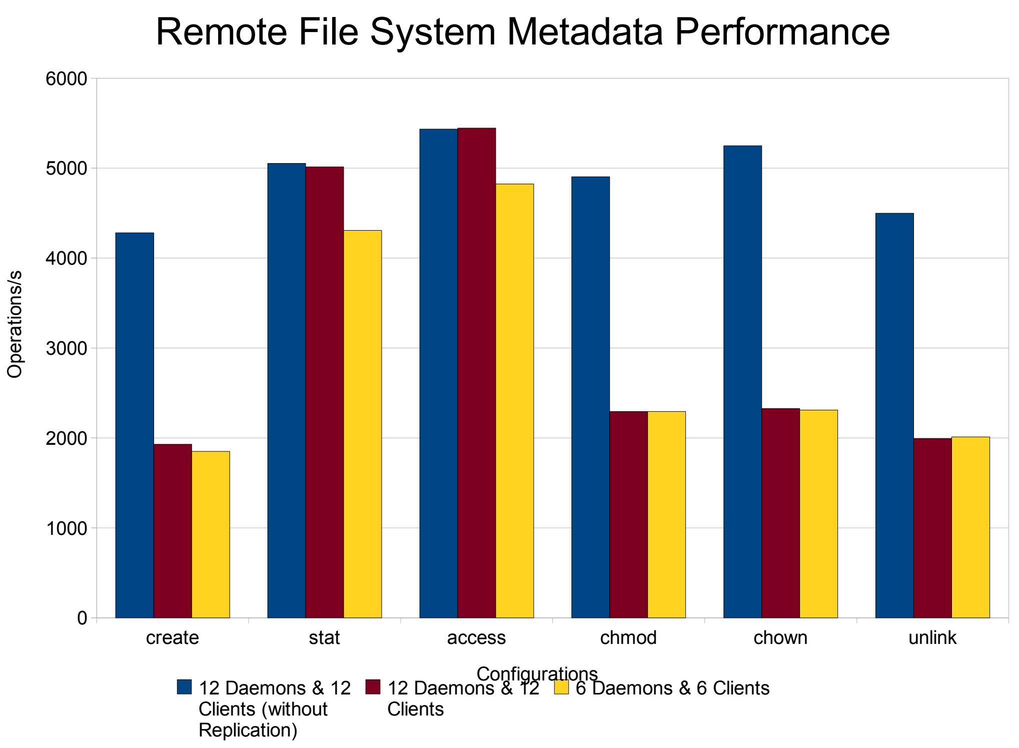
Operation	MB/s
`write`	14
`read`	1,700

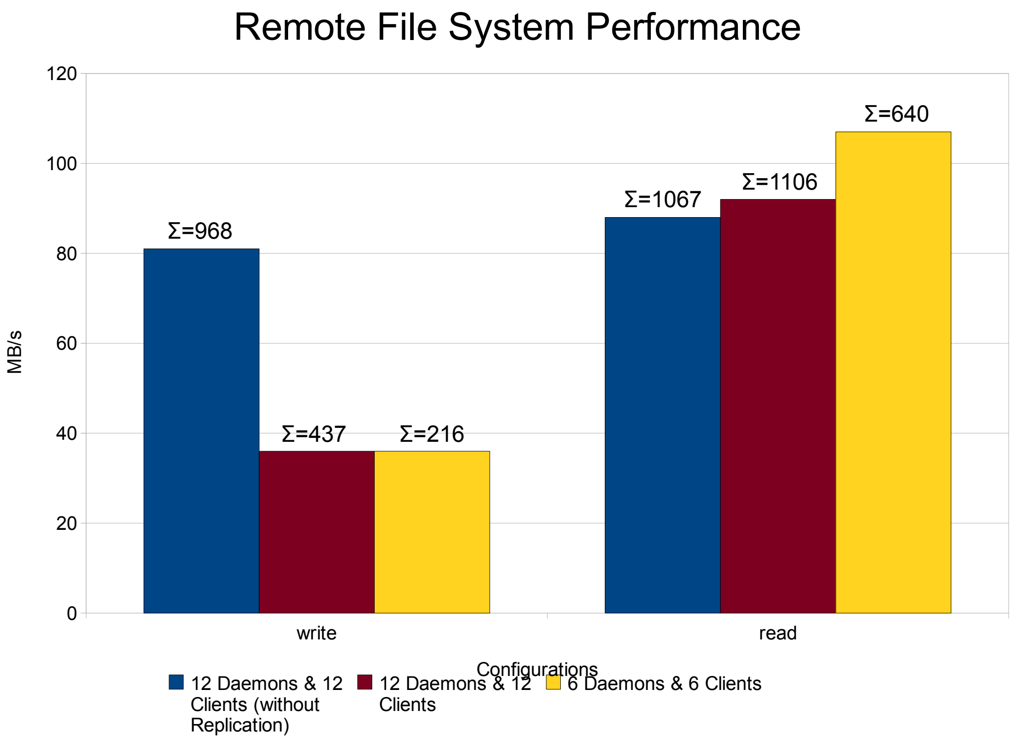
Direct I/O

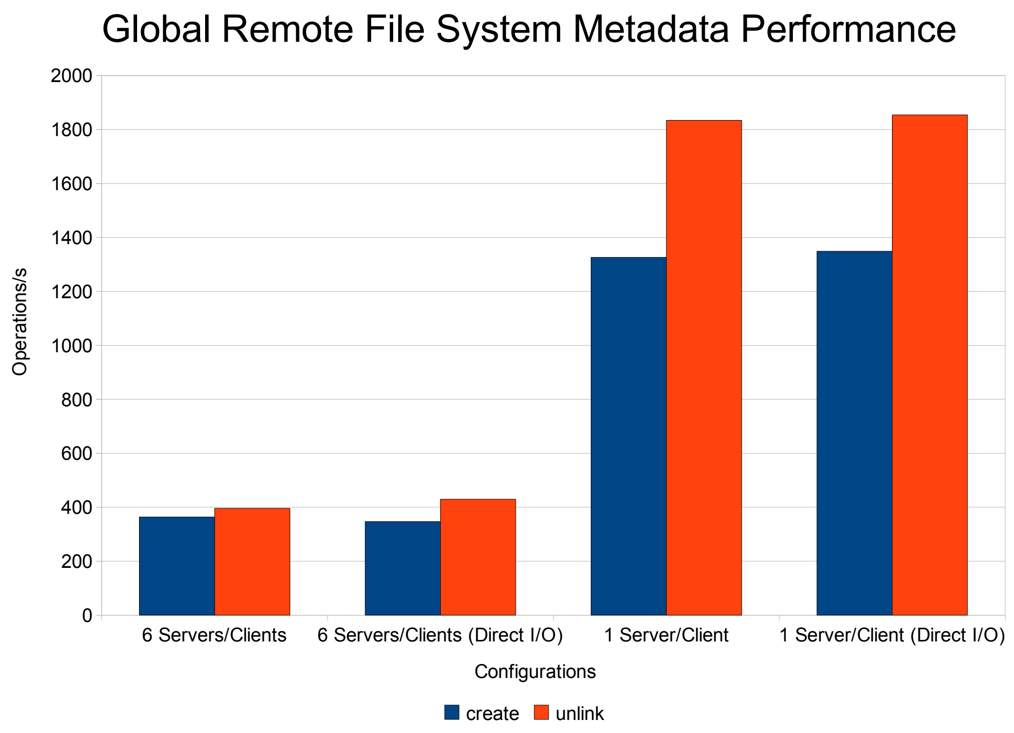
Operation	Ops/s
`create`	1,349
`unlink`	1,854

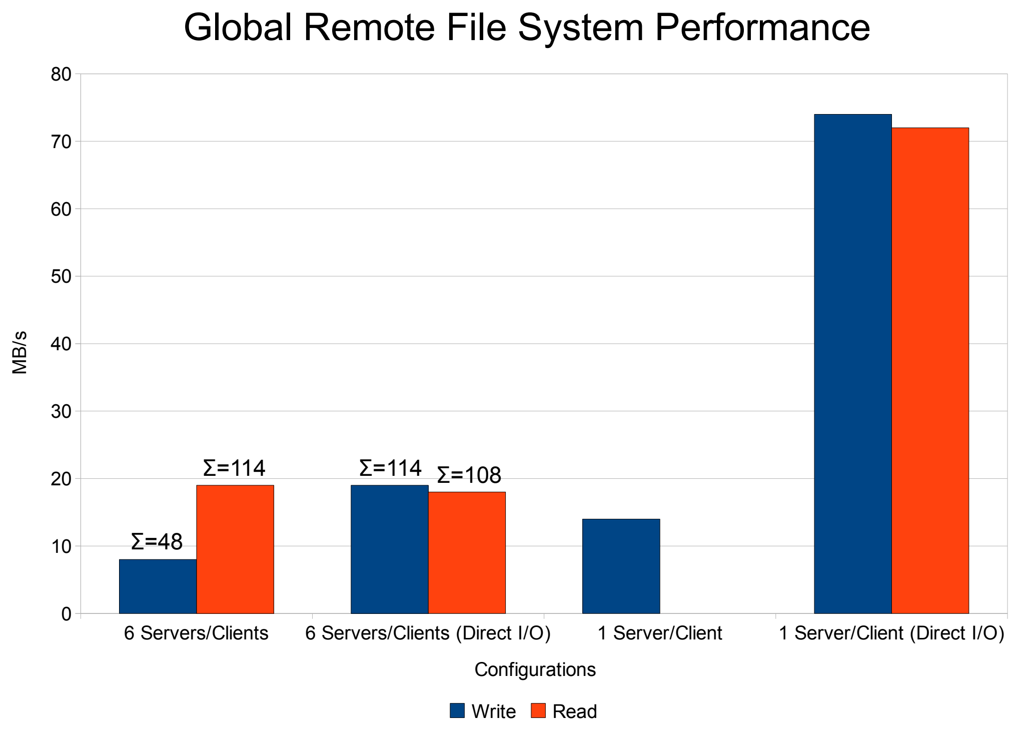
Operation	MB/s
`write`	74
`read`	72

Figures


Remote File System Metadata Performance
Operations/s
6000
5000
4000
3000
2000
1000
0
create
stat
access
chmod
chown
unlink
Configurations
12 Daemons & 12 Clients (without Replication)
12 Daemons & 12 Clients
6 Daemons & 6 Clients


Remote File System Performance
MB/s
120
100
80
60
40
20
0
Σ=968
Σ=437
Σ=216
Σ=1067
Σ=1106
Σ=640
write
read
Configurations
12 Daemons & 12 Clients (without Replication)
12 Daemons & 12 Clients
6 Daemons & 6 Clients


Global Remote File System Metadata Performance
Operations/s
2000
1800
1600
1400
1200
1000
800
600
400
200
0
6 Servers/Clients
6 Servers/Clients (Direct I/O)
1 Server/Client
1 Server/Client (Direct I/O)
Configurations
create
unlink


Global Remote File System Performance
MB/s
80
70
60
50
40
30
20
10
0
Σ=48
Σ=114
Σ=114
Σ=108
6 Servers/Clients
6 Servers/Clients (Direct I/O)
1 Server/Client
1 Server/Client (Direct I/O)
Configurations
Write
Read

7. Conclusion

7.1. Global Remote File System

The Global Remote File System's performance is mainly limited by FUSE. FUSE uses read and write buffers that are at most 128 KB in size. When using normal I/O, read buffers of size 128 KB and write buffers of size 4 KB are used. When using direct I/O, read buffers of size 128 KB and write buffers of size 128 KB are used. However, direct I/O disables all caching. The FUSE 2.8.0 pre-release supports write buffers of up to 512 KB.

A. Usage Instructions

A.1. Installing Required Packages

```
$ sudo aptitude install libfuse-dev libglib2.0-dev
```

A.2. Compiling Everything

```
$ cd rfs
$ make
```

A.3. Remote File System Daemon

```
$ cd rfs/rfsd
$ ./rfsd --prefix=/tmp
```

A.4. Remote File System Client

```
$ cd rfs/rfsc
$ ./rfsc host:port password
```

A.5. Global Remote File System

```
$ cd rfs/grfs
$ ./grfs host1:port1 host2:port2 ${MOUNT_POINT}
```

A.5.1. Debugging

To debug the FUSE file system, use the `-f` argument.

```
$ cd rfs/grfs
$ gdb --args ./grfs -f host1:port1 host2:port2 ${MOUNT_POINT}
```

The `-d` argument causes FUSE to print debug output. Do **not** run benchmarks with this.

```
$ cd rfs/grfs
$ ./grfs -d host1:port1 host2:port2 ${MOUNT_POINT}
```

B. `screen` Configuration Files

The following files can be used like follows.

```
$ screen -c config_file
```

B.1. Starting Remote File System Daemons

B.1.1. Without Replication

```
startup_message off

screen  0 ssh root@vqstor01 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.51 --port=8888 --daemon --prefix=/dev/shm/rfs'

screen  1 ssh root@vqstor01 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.51 --port=6666 --daemon --prefix=/mnt/master'
screen  2 ssh root@vqstor02 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.52 --port=6666 --daemon --prefix=/mnt/master'
screen  3 ssh root@vqstor03 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.53 --port=6666 --daemon --prefix=/mnt/master'
screen  4 ssh root@vqstor04 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.54 --port=6666 --daemon --prefix=/mnt/master'
screen  5 ssh root@vqstor05 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.55 --port=6666 --daemon --prefix=/mnt/master'
screen  6 ssh root@vqstor06 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.56 --port=6666 --daemon --prefix=/mnt/master'
screen  7 ssh root@vqstor07 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.57 --port=6666 --daemon --prefix=/mnt/master'
screen  8 ssh root@vqstor08 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.58 --port=6666 --daemon --prefix=/mnt/master'
screen  9 ssh root@vqstor09 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.59 --port=6666 --daemon --prefix=/mnt/master'
screen 10 ssh root@vqstor10 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.60 --port=6666 --daemon --prefix=/mnt/master'
screen 11 ssh root@vqstor11 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.61 --port=6666 --daemon --prefix=/mnt/master'
screen 12 ssh root@vqstor12 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.62 --port=6666 --daemon --prefix=/mnt/master'
```

B.1.2. With Replication

```
startup_message off

screen  0 ssh root@vqstor01 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --bind=10.0.0.51 --port=8888 --daemon --prefix=/dev/shm/rfs'

screen  1 ssh root@vqstor01 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.51 --port=7777 --daemon --prefix=/mnt/slave'
screen  2 ssh root@vqstor02 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.52 --port=7777 --daemon --prefix=/mnt/slave'
screen  3 ssh root@vqstor03 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.53 --port=7777 --daemon --prefix=/mnt/slave'
screen  4 ssh root@vqstor04 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.54 --port=7777 --daemon --prefix=/mnt/slave'
screen  5 ssh root@vqstor05 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.55 --port=7777 --daemon --prefix=/mnt/slave'
screen  6 ssh root@vqstor06 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.56 --port=7777 --daemon --prefix=/mnt/slave'
screen  7 ssh root@vqstor07 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.57 --port=7777 --daemon --prefix=/mnt/slave'
screen  8 ssh root@vqstor08 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.58 --port=7777 --daemon --prefix=/mnt/slave'
screen  9 ssh root@vqstor09 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.59 --port=7777 --daemon --prefix=/mnt/slave'
screen 10 ssh root@vqstor10 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.60 --port=7777 --daemon --prefix=/mnt/slave'
screen 11 ssh root@vqstor11 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.61 --port=7777 --daemon --prefix=/mnt/slave'
screen 12 ssh root@vqstor12 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --slave --bind=10.0.1.62 --port=7777 --daemon --prefix=/mnt/slave'

screen 13 ssh root@vqstor01 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.52:7777 --bind=10.0.0.51 --port=6666 --daemon --prefix=/mnt/master'
screen 14 ssh root@vqstor02 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.51:7777 --bind=10.0.0.52 --port=6666 --daemon --prefix=/mnt/master'
screen 15 ssh root@vqstor03 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.54:7777 --bind=10.0.0.53 --port=6666 --daemon --prefix=/mnt/master'
screen 16 ssh root@vqstor04 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.53:7777 --bind=10.0.0.54 --port=6666 --daemon --prefix=/mnt/master'
screen 17 ssh root@vqstor05 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.56:7777 --bind=10.0.0.55 --port=6666 --daemon --prefix=/mnt/master'
screen 18 ssh root@vqstor06 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.55:7777 --bind=10.0.0.56 --port=6666 --daemon --prefix=/mnt/master'
screen 19 ssh root@vqstor07 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.58:7777 --bind=10.0.0.57 --port=6666 --daemon --prefix=/mnt/master'
screen 20 ssh root@vqstor08 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.57:7777 --bind=10.0.0.58 --port=6666 --daemon --prefix=/mnt/master'
screen 21 ssh root@vqstor09 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.60:7777 --bind=10.0.0.59 --port=6666 --daemon --prefix=/mnt/master'
screen 22 ssh root@vqstor10 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.59:7777 --bind=10.0.0.60 --port=6666 --daemon --prefix=/mnt/master'
screen 23 ssh root@vqstor11 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd --master=10.0.1.62:7777 --bind=10.0.0.61 --port=6666 --daemon
```

```
    ↪ --prefix=/mnt/master'
screen 24 ssh root@vqstor12 'LD_LIBRARY_PATH=/root/lib /root/bin/rfsd
    ↪ --master=10.0.1.61:7777 --bind=10.0.0.62 --port=6666 --daemon
    ↪ --prefix=/mnt/master'
```

B.2. Starting Remote File System Clients

```
startup_message off

screen  1 ssh root@vqcomp01 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.62:6666 random
    ↪ /vqstor12 | tee vqcomp01.txt'
screen  2 ssh root@vqcomp02 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.51:6666 random
    ↪ /vqstor01 | tee vqcomp02.txt'
screen  3 ssh root@vqcomp03 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.52:6666 random
    ↪ /vqstor02 | tee vqcomp03.txt'
screen  4 ssh root@vqcomp04 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.53:6666 random
    ↪ /vqstor03 | tee vqcomp04.txt'
screen  5 ssh root@vqcomp05 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.54:6666 random
    ↪ /vqstor04 | tee vqcomp05.txt'
screen  6 ssh root@vqcomp06 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.55:6666 random
    ↪ /vqstor05 | tee vqcomp06.txt'
screen  7 ssh root@vqcomp07 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.56:6666 random
    ↪ /vqstor06 | tee vqcomp07.txt'
screen  8 ssh root@vqcomp08 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.57:6666 random
    ↪ /vqstor07 | tee vqcomp08.txt'
screen  9 ssh root@vqcomp09 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.58:6666 random
    ↪ /vqstor08 | tee vqcomp09.txt'
screen 10 ssh root@vqcomp10 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.59:6666 random
    ↪ /vqstor09 | tee vqcomp10.txt'
screen 11 ssh root@vqcomp11 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.60:6666 random
    ↪ /vqstor10 | tee vqcomp11.txt'
screen 12 ssh root@vqcomp12 'LD_LIBRARY_PATH=/root/lib
    ↪ RFSC_BARRIER=10.0.0.51:8888:random /root/bin/rfsc 10.0.0.61:6666 random
    ↪ /vqstor11 | tee vqcomp12.txt'
```