Project report

“Replacement of SCST iSCSI driver with in-kernel Linux-IO driver”

Prepared by
Denis Koptev,
Vladislav Kulakovsky

Peter the Great St. Petersburg Polytechnic University
Institute of Computer Science and Technology
High School of Software Engineering

Supervisor is
Yuri Stotski

Dell EMC

Date of report __.__.2017

1. Contents

2. Introduction

iSCSI is an acronym for Internet Small Computer Systems Interface, an Internet Protocol (IP)-based storage networking standard for linking data storage facilities. It provides block-level access to storage devices by carrying SCSI commands over a TCP/IP network. iSCSI is used to facilitate data transfers over intranets and to manage storage over long distances. It can be used to transmit data over local area networks (LANs), wide area networks (WANs), or the Internet and can enable location-independent data storage and retrieval.

The protocol allows clients (called initiators) to send SCSI commands (CDBs) to storage devices (targets) on remote servers. It is a storage area network (SAN) protocol, allowing organizations to consolidate storage into storage arrays while providing clients (such as database and web servers) with the illusion of locally attached SCSI disks. It mainly competes with Fibre Channel, but unlike traditional Fibre Channel which usually requires dedicated cabling, iSCSI can be run over long distances using existing network infrastructure.

Dell EMC has its storage systems run under Linux OS. So, there are some implementations of iSCSI target and initiator (server and client) for Linux. We will consider such non-commercial open source implementations as SCST and Linux-IO.

At the moment Dell EMC uses SCST driver. However, Linux-IO driver has recently been included in the Linux kernel as a standard one. As a result, it is reasonable to assume that LIO can have better performance than SCST. In addition, it is always presented in the kernel and does not require an additional installation.

2.1. Basic goal

The main purpose of the project is to study the possibility of replacement of the current iSCSI SCST driver in Linux-based input-output subsystem with a more up-to-date in-kernel Linux-IO driver.

2.2. Tasks of the project

1. Make an overview of Linux-IO driver and compare its capabilities with SCST driver.
2. Develop the possibility of replacement of SCST driver with an up-to-date version of Linux-IO in the high capacity data transfer system with placement of the block devices at the user space level.
3. Explore and produce the optimization of capacity.

3. Works done

3.1. Overview of Linux-IO driver

In essence, iSCSI allows two hosts to negotiate and then exchange SCSI commands using Internet Protocol (IP) networks.

An initiator functions as an iSCSI client. An initiator typically serves the same purpose to a computer as a SCSI bus adapter would, except that, instead of physically cabling SCSI devices (like hard drives and tape changers), an iSCSI initiator sends SCSI commands over an IP network.

The iSCSI specification refers to a storage resource located on an iSCSI server (more generally, one of potentially many instances of iSCSI storage nodes running on that server) as a target.

An iSCSI target is often a dedicated network-connected hard disk storage device, but may also be a general-purpose computer, since as with initiators, software to provide an iSCSI target is available for most mainstream operating systems.

SCSI target is the endpoint that does not initiate sessions, but instead waits for initiators' commands and provides required input/output data transfers. The target usually provides to the initiators one or more LUNs, because otherwise no read or write command would be possible.

Linux-IO (LIO) Target is an open-source implementation of the SCSI target that has become the standard one included in the Linux kernel.

LIO is maintained by Datera, Inc., a Silicon Valley vendor of storage systems and software. On January 15, 2011, LIO SCSI target engine was merged into the Linux kernel mainline, in kernel version 2.6.38, which was released on March 14, 2011. Additional fabric modules have been merged into subsequent Linux releases.

The LIO Linux SCSI Target implements a generic SCSI target that provides remote access to most data storage device types over all prevalent storage fabrics and protocols. LIO neither directly accesses data nor does it directly communicate with applications. LIO provides a highly efficient, fabric-independent and fabric-transparent abstraction for the semantics of numerous data storage device types.

Now, we can consider the main entities of the iSCSI target represented by the Linux-IO driver.

3.1.1. Backstores

LIO modularizes the actual data storage. These are referred to as "backstores" or "storage engines". The target comes with backstores that allow a file, a block device, RAM, or another SCSI device to be used for the local storage needed for the exported SCSI LUN. Like the rest of LIO, these are implemented entirely as kernel code.

Backstores provide the SCSI target with generalized access to data storage devices by importing them via corresponding device drivers. Backstores don't need to be physical SCSI devices.

The most important backstore media types are:

- Block: The block driver allows using raw Linux block devices as backstores for export via LIO. This includes physical devices, such as HDDs, SSDs, CDs/DVDs, RAM disks, etc., and logical devices, such as software or hardware RAID volumes or LVM volumes.

- File: The file driver allows using files that can reside in any Linux file system or clustered file system as backstores for export via LIO.
- Raw: The raw driver allows using unstructured memory as backstores for export via LIO.

3.1.2. iSCSI

The Internet Small Computer System Interface (iSCSI) fabric module allows the transport of SCSI traffic across standard IP networks. Fabric modules implement the frontend of the SCSI target by encapsulating and abstracting the properties of the various supported interconnect. The following fabric modules are available.

3.1.3. Targetcli

Targetcli is a user space single-node management command line interface (CLI) for LIO. It supports all fabric modules and is based on a modular, extensible architecture, with plug-in modules for additional fabric modules or functionality.

Targetcli is implemented in Python and consists of three main modules:

- the underlying rtslib and API.
- the configshell, which encapsulates the fabric-specific attributes in corresponding 'spec' files.
- the targetcli shell itself.

Interface represented by targetcli in Linux terminal:

```
o- / ......................................................................... [...]
  o- backstores .............................................................. [...]
  | o- block .................................... [Storage Objects: 0]
  | o- fileio ................................... [Storage Objects: 0]
  | o- pscsi .................................... [Storage Objects: 0]
  | o- ramdisk .................................. [Storage Objects: 0]
  o- iscsi ................................................ [Targets: 0]
  o- loopback ............................................. [Targets: 0]
  o- vhost ................................................ [Targets: 0]
```

Targetcli can be installed from almost all Linux distributions' repositories in case it is not already presented.

All storage types in backstores are supported by LIO from the kernel space. Configuration of target and devices is stored and processed via sysfs – virtual filesystem in Linux. Sysfs exports information about devices and drivers presented in the system to the user space.

3.2. Comparison with SCST driver

A competing generic SCSI target module for Linux is SCST. SCST is now used by Dell EMC because it provides an ability to configure command processing easily from user space (literally it works not only in kernel space). On the other hand, Linux-IO is an entirely in-kernel driver.

SCST consists of three groups of modules:

- The SCST core, a protocol-independent engine for processing SCSI commands.
- Target drivers which receive SCSI commands from a SCSI initiator, pass these SCSI to the SCST core and send back replies to the initiator.

- Storage drivers a.k.a. device handlers which interact with the storage medium. The supported local storage interfaces are SCSI, block device, file and scst_user. scst_user is an SCST-specific protocol that allows efficient implementation of storage drivers in user space.

Like in the LIO driver configuration of all these modules happens via a sysfs interface. Although direct configuration of SCST via its sysfs interface is convenient, the tool called scstadmin allows to control SCST via its sysfs interface and also to save and restore the SCST configuration.

One new use case that other non-kernel target solutions, such as tgt, are able to support is using Gluster's GLFS or Ceph's RBD as a backstore. The target then serves as a translator, allowing initiators to store data in these non-traditional networked storage systems, while still only using standard protocols themselves.

If the target is a userspace process, supporting these is easy. tgt, for example, needs only a small adapter module for each, because the modules just use the available userspace libraries for RBD and GLFS.

Adding support for these backstores in LIO is considerably more difficult, because LIO is entirely kernel code.

3.3. Possibility of replacement of SCST driver with Linux-IO driver

SCST is now using by Dell EMC in its products. This driver as it was said has a module that supports the interaction with user space (SCST-specific protocol).

The Linux user space provides several advantages for applications, including more robust and flexible process management, standardized system-call interface, simpler resource management, a large number of libraries for XML, and regular expression parsing, among others. It also makes applications more straightforward to debug by providing memory isolation and independent restart.

On the other hand, Linux-IO doesn’t have user space modules in its standard configuration (i.e.: via targetcli). Almost all other features concerning the configuration of target are the same.

So, the task of replacement is divided into several sub-tasks:

1. create some standard configurations of LIO using targetcli;
2. explore this configuration in sysfs;
3. create the facilities (scripts) for custom configuration without CLI;
4. learn whether there are modules for user space interconnection available for LIO;
5. adopt the scripts using found modules and information about how this new configuration looks like in sysfs;
6. create the simplest SCSI command handler in user space.

3.3.1. Creating the standard LIO configuration via targetcli interface

All configuration for target and initiator will be created on one virtual machine (locally).

We will be using targetcli (a user space single-node management command line interface for LIO). If it is not present in a system, it can be installed simply: `apt-get install targetcli`.

At first, we will create a standard configuration for fileio backstore (LIO will be working with a file as a device). First, we will have zero configuration in targetcli without targets and devices set up. This configuration can be seen via usual `ls` command:

```
o- / ..................................................................... [...]
  o- backstores ........................................................... [...]
  | o- block ..................................... [Storage Objects: 0]
  | o- fileio .................................... [Storage Objects: 0]
  | o- pscsi ..................................... [Storage Objects: 0]
  | o- ramdisk ................................... [Storage Objects: 0]
  o- iscsi ..................................................... [Targets: 0]
  o- loopback .................................................. [Targets: 0]
  o- vhost ..................................................... [Targets: 0]
```

To create a backstore corresponding to work with files we must create a file itself and move to /backstore/fileio directory, create a new device and set some configuration parameters:

In any directory create a file, i.e.: `/home > touch tgt_file`

In targetcli:

```
> cd /backstores/fileio
> create file_dev /home/tgt_file
```

As a result, our configuration from the root of targetcli will look like this:

```
o- / ..................................................................... [...]
  o- backstores ........................................................... [...]
  | o- block ..................................... [Storage Objects: 0]
  | o- fileio .................................... [Storage Objects: 1]
  | | o- file_dev  [/home/denis/Desktop/tgt_file (1.0KiB)]
  | o- pscsi ..................................... [Storage Objects: 0]
  | o- ramdisk ................................... [Storage Objects: 0]
  o- iscsi ..................................................... [Targets: 0]
  o- loopback .................................................. [Targets: 0]
  o- vhost ..................................................... [Targets: 0]
```

Then we need to set the target configuration, in the other words – create target:

```
> cd iscsi
> create iqn.2017-07.com.test:target-10-15-33
```

The IQN format takes the form *iqn.yyyy-mm.naming-authority:unique*:

- yyyy-mm is the year and month when the naming authority was established.
- naming-authority is usually reverse syntax of the Internet domain name of the naming authority. For example, the iscsi.vmware.com naming authority could have the iSCSI qualified name form of iqn.1998-01.com.vmware.iscsi.
- unique name is any name you want to use, for example, the name of your host. The naming authority must make sure that any names assigned following the colon are unique

The configuration will change in the following manner:

```
o- / ..................................................................... [...]
  o- backstores ........................................................... [...]
  | o- block ..................................... [Storage Objects: 0]
  | o- fileio .................................... [Storage Objects: 1]
  | | o- file_dev  [/home/denis/Desktop/tgt_file (1.0KiB) write-thru act]
  | o- pscsi ..................................... [Storage Objects: 0]
  | o- ramdisk ................................... [Storage Objects: 0]
  o- iscsi ..................................................... [Targets: 1]
  | o- iqn.2017-07.com.test:target-10-15-33 ................... [TPGs: 1]
  |   o- tpg1 ................................... [no-gen-acls, no-auth]
  |     o- acls ..................................................... [ACLs: 0]
```

```
    |     o- luns ..................................................... [LUNs: 0]
    |     o- portals ................................................... [Portals: 0]
    o- loopback ..................................................... [Targets: 0]
    o- vhost ........................................................ [Targets: 0]
```

The target portal group was created. Moreover, in our example we will set authorization to None for the brevity of testing: `/iscsi> set discovery_auth enable=0`

Now we must create LUN in the target for our file device in backstores:

```
> cd /iscsi/iqn…/tpg1/luns

> create /backstores/fileio/file_dev
```

Then we need to add initiator in the access command list in target for security and access reasons. Here we use create command and specify the initiator's iqn. Initiator's iqn is stored locally on the initiator machine: */etc/iscsi/initiatorname.iscsi*. In this example:

```
> cd ../acls
> create iqn.1993-08.org.debian:01:ef2e26bf3a9e
```

Now we can create a portal (standard 0.0.0.0:3260):

```
> cd ../portals
> create
```

So, the final configuration is:

```
o- / ............................................................... [...]
  o- backstores .................................................... [...]
  | o- block ...................................... [Storage Objects: 0]
  | o- fileio ..................................... [Storage Objects: 1]
  | | o- file_dev  [/home/denis/Desktop/tgt_file (1.0KiB) write-thru act]
  | o- pscsi ...................................... [Storage Objects: 0]
  | o- ramdisk .................................... [Storage Objects: 0]
  o- iscsi ................................................ [Targets: 1]
  | o- iqn.2017-07.com.test:target-10-15-33 ................... [TPGs: 1]
  |   o- tpg1 ................................... [no-gen-acls, no-auth]
  |     o- acls ............................................... [ACLs: 1]
  |     | o- iqn.1993-08.org.debian:01:ef2e26bf3a9e .... [Mapped LUNs: 1]
  |     |   o- mapped_lun0 ................. [lun0 fileio/file_dev (rw)]
  |     o- luns ............................................... [LUNs: 1]
  |     | o- lun0 ...... [fileio/file_dev (/home/denis/Desktop/tgt_file)]
  |     o- portals ......................................... [Portals: 1]
  |       o- 0.0.0.0:3260 ........................................... [OK]
  o- loopback ............................................. [Targets: 0]
  o- vhost ................................................ [Targets: 0]
```

Now we can leave targetcli (`exit`). By default, all configuration will be saved automatically.

3.3.2. Setting up the initiator side

Open-iSCSI package for Linux will be used here. It can be installed in the same manner that targetcli.

First, we discover all available targets on specified port:

```
> iscsiadm -m discovery -t sendtargets -p 127.0.0.1
127.0.0.1:3260,1 iqn.2017-07.com.test:target-10-15-33
```

We can see our target from initiator's side. Now we can connect to it:

```
> iscsiadm -m node -T iqn.2017-07.com.test:target-10-15-33 -p 127.0.0.1 –login
Logging in to [iface: default, target: iqn.2017-07.com.test:target-10-15-33, portal: 127.0.0.1,3260] (multiple)
Login to [iface: default, target: iqn.2017-07.com.test:target-10-15-33, portal: 127.0.0.1,3260] successful.
```

We can list iSCSI devices via lsscsi (package can be easily installed) and find our device:

```
> lsscsi | grep FILEIO
[3:0:0:0]    disk    LIO-ORG  FILEIO           4.0   /dev/sdb
```

Now all read and write operations are supported. For example, we can write or read in file using `dd` command.

3.3.3. Configurating a block device

Steps to create a configuration with block device instead of file device are similar with a few differences:

First, we need to create backstore for block device:

```
> backstores/iblock create name=block dev=/dev/sda2
```

In this example, we chose sda2 partition as a device representation. Also, we can specify block size and other parameters in configuration.

Target creation and configuration is equivalent: we need to create lun (set the link to block backstore) and then add appropriate initiator to the access command list, create a portal and connect to the target via iscsiadm tool.

3.3.4. Review of the configuration in sysfs

After all operations in targetcli we can explore changes in sysfs. Here we will display some sysfs changes found (others are omitted for brevity).

1. iscsi target configuration appears here: `/sys/kernel/config/target/iscsi/iqn...`

2. In this folder:

```
drwxr-xr-x 7 root root 0 Jun 22 15:55 fabric_statistics
drwxr-xr-x 8 root root 0 Jun 22 16:12 tpgt_1
```

3. In tpgt_1:

```
drwxr-xr-x 3 root root    0 Jun 22 15:56 acls
drwxr-xr-x 2 root root    0 Jun 22 16:03 attrib
drwxr-xr-x 2 root root    0 Jun 22 16:03 auth
-r--r--r-- 1 root root 4096 Jun 22 16:14 dynamic_sessions
-rw-r--r-- 1 root root 4096 Jun 22 15:55 enable (1)
drwxr-xr-x 3 root root    0 Jun 22 15:56 lun
drwxr-xr-x 3 root root    0 Jun 22 15:55 np
drwxr-xr-x 2 root root    0 Jun 22 16:03 param
```

4. In acls folder:

```
drwxr-xr-x 7 root root 0 Jun 22 16:03 iqn.1993-08.org.debian:01:ef2e26bf3a9e
```

5. In lun folder: `lun_0`

6. In lun_0 folder:

```
    -rw-r--r-- 1 root root 4096 Jun 22 16:22 alua_tg_pt_gp
    -rw-r--r-- 1 root root 4096 Jun 22 16:22 alua_tg_pt_offline
    -rw-r--r-- 1 root root 4096 Jun 22 16:22 alua_tg_pt_status
    -rw-r--r-- 1 root root 4096 Jun 22 16:22 alua_tg_pt_write_md
    lrwxrwxrwx   1   root   root            0   Jun   22   15:56   bbfb398a4c   ->
../../../../../../target/core/fileio_0/test_dev
    drwxr-xr-x 5 root root    0 Jun 22 15:56 statistics
```

Considering the device creation, we tracked following changes:

In `/sys/kernel/config/target/core` appears `fileio_0/file_dev`

In this directory, we have some configuration parameters. One of them is **"**`enable`**"** which has become 1 (device is enabled since target is connected to it).

As we can see, all our actions in targetcli had an influence on sysfs. The mechanism is that targetcli (written in Python) uses some scripts and creates different entities for different purposes. For example, it creates target iqn folder in sysfs when we call `create` command. The kernel then assigns the responsibility to the driver. It is logical to assume, that we can process all this actions by hands in the Linux terminal or write our own scripts.

3.3.5. Scripts for custom configuration without targetcli

With all information described in the previous paragraph we can start writing bash scripts to automate configuration processes. In a few words, we have all changes tracked for sysfs configuration. We can do all these changes manually, creating "directories" (sysfs entities in fact) and files (sysfs configuration parameters). Moreover, our goal is to create custom configuration for LIO in order to let it work with user space. So, at the first step we will create scripts for the standard configuration and then we will try to customize them. As a result, these scripts will provide us with an ability to transform the configuration for different purposes.

We started with the script for target configuration: `target_create.sh`. This script doesn't require any arguments. It generates IQN from current date and time, creates standard portal, adds initiator IQN (for local configuration) and turns off authorization. Script can be found in *6.1. Annex 1: target_create* script.

Then we need a script to create backstore file devices: `file_create.sh`. Script requires 3 arguments: a full path to a file, device name and size. It can be found in *6.2. Annex 2: file_create script*.

Launching these scripts, we will be able to create the same configuration as it was created previously using targetcli.

Script for the creation of a block backstore: `block_create.sh`. It is a bit more complicated. It requires 2 arguments: path to a block device and device name. Script can be found in *6.3. Annex 3: block_create* script.

Then we need to set up created target for work with created devices: `target_setup.sh.` It requires target IQN and backstore path. This script just creates a lun in order to connect the target with the backstore. It can be found in *6.4. Annex 4: target_setup script*.

3.3.6. Usage of scripts

Using created scripts, we can easily configure LIO to work with a block device. Script `block_create.sh` creates logical volume and backstore. Scripts `target_create.sh` and `target_setup.sh` create target, tpg, acl and lun.

Device creation

1. Create directory in which we'll store file for log volume

```
> mkdir /home/denis/iscsi_files
```

2. Find block_create script. This script should be run as root.

3. Running this script, you need to specify directory and name for file and name for backstore device. In this example:

```
> sh block_create.sh /home/iscsi_files/test_file test_block
```

Script will give you some info:

```
Physical volume "/dev/loop2" successfully created. Volume group "vol_group_1" successfully created Logical volume "log_vol" created. Your backstore path: iblock_0/test_block
```

`/dev/loop2` is the first loopback device which is free (in this example). You will use backstore path (here: `iblock_0/test_block`) later.

Target setup

1. Locate target_create.sh script and run it:

```
> sh target_create.sh
```

1. Locate target_setup.sh script. You have to specify backstore path, given to you by block_create script. Script will automatically generate iqn, you don't need to specify it.

```
> sh target_setup.sh <iqn...> iblock_0/test_block
```

2. We can check this configuration via targetcli

Initiator setup

Discover targets:

```
> iscsiadm -m discovery -t sendtargets -p 0.0.0.0
127.0.0.1:3260,1 iqn.2017-06.com.test:target-25-04-40
```

Connect:

```
> iscsiadm -m node -p 127.0.0.1 --login
```

Check:

```
> iscsiadm -m session -P 0
```

Or use `lsscsi` command to find our new device. In this example - `/dev/sdb.`

Log out of session:

```
> iscsiadm -m node -u
```

3.3.7. Standard module for user space interconnection

In the previous paragraph we created scripts, which work with sysfs and let us to set the custom configuration of LIO.

Now we need to find a tool for user space interconnection. It was revealed, that LIO driver has a special driver module for this goal. It is called TCMU. By the way, TCM is the new name for LIO driver modules.

TCMU (TCM in user space) is a package that connects to the TCM in user space kernel module, a part of the LIO stack. TCMU allows user space programs to be written which act as iSCSI targets. The core of the TCMU interface is a memory region that is shared between kernel and userspace.

TCMU uses the pre-existing UIO subsystem. UIO allows device driver development in userspace, and this is conceptually very close to the TCMU use case, except instead of a physical device, TCMU implements a memory-mapped layout designed for SCSI commands. Using UIO also benefits TCMU by handling device introspection (e.g. a way for userspace to determine how large the shared region is) and signaling mechanisms in both directions.

There are no embedded pointers in the memory region. Everything is expressed as an offset from the region's starting address. This allows the ring to still work if the user process dies and is restarted with the region mapped at a different virtual address.

Module can be found via `lsmod` command. So, this module should obviously be loaded to the kernel in order to start the interconnection with user space.

```
/# lsmod | grep target
iscsi_target_mod      290816  1
target_core_mod       352256  2 iscsi_target_mod
configfs               40960  4 rdma_cm,iscsi_target_mod,target_core_mod
scsi_mod              225280  14 ib_iser,sd_mod,…

/lib/modules/4.9…amd64/kernel/drivers/target# modprobe target_core_user
/# lsmod | grep target
target_core_user       20480  0
uio                    16384  1 target_core_user
iscsi_target_mod      290816  1
target_core_mod       352256  3 iscsi_target_mod,target_core_user
configfs               40960  4
```

3.3.8. tcmu-runner daemon

The TCMU userspace-passthrough backstore allows a userspace process to handle requests to a LUN. But since the kernel-user interface that TCMU provides must be fast and flexible, it is complex enough that we'd like to avoid each userspace handler having to write boilerplate code. As a result, we have found an open-source project on GitHub called tcmu-runner.

tcmu-runner handles the messy details of the TCMU interface and exports a friendlier C plugin module API. Modules using this API are called "TCMU handlers". Handler authors can write code just to handle the SCSI commands as desired, and can also link with whatever userspace libraries they like.

Daemon can be cloned from GitHub repository (https://github.com/open-iscsi/tcmu-runner) and built following a simple instruction:

```
> git clone https://github.com/open-iscsi/tcmu-runner.git
```

Installing dependencies (tested on Ubuntu 16.04):

1. Install cmake: `sudo apt-get install cmake`
2. Install other dependent libraries:

```
> apt-get install libnl-3-dev
> apt-get install libglib2.0-dev
> apt-get install libpthread-*
> apt-get install libdlib-dev
> apt-get install libkmod-dev
> apt-get install glusterfs-*
> apt-get install librbd-dev
> apt-get install zlib*
```

Building sources:

```
> cd tcmu-runner
> cmake .
```

You can have errors like this:

```
Please set them or make sure they are set and tested correctly in the CMake files: LIBNL_GENL_LIB linked by target "tcmu" in directory /root/tcmu-runner
```

It means you have to install one more lib:

```
> apt-get install libnl-genl-3-dev
```

Finishing build:

```
> make
```

Ensure, that `target_core_user` module is loaded (`lsmod`). If not, load it from `/lib/modules/…/kernel/drivers/target`. Otherwise, tcmu-runner should load it.

Copy `tcmu-runner.conf` to `/etc/dbus-1/system.d/.` This allows tcmu-runner to be on the system bus, which is privileged. Copy tcmu-runner executable from tcmu-runner folder to `/usr/bin/.` Create folder `/etc/tcmu/.` Copy `tcmu-runner/tcmu.conf` to `/etc/tcmu/.`

Launching and getting help:

```
> ./tcmu-runner -h
usage:
    tcmu-runner [options]
options:
    -h, --help: print this message and exit
    -V, --version: print version and exit
    -d, --debug: enable debug messages
    --handler-path: set path to search for handler modules
        default is /usr/local/lib/tcmu-runner
    -l, --tcmu-log-dir: tcmu log dir
```

By default, tcmu-runner has its own example handlers, which demonstrate how to work with rdb (RADOS Block Devices), qcow (QUEMU Copy on Write) and glfs (Gluster) tools. Running the daemon, we can specify tcmu-runner folder as a folder for handlers doing following:

```
> ./tcmu-runner --handler-path <path-to-daemon>
```

Now we can use targetcli to check that everything works. Backstores folder should have the following device types:

```
o- / ..................................................................... [...]
  o- backstores ......................................................... [...]
  | o- block ............................................ [Storage Objects: 0]
  | o- fileio ........................................... [Storage Objects: 0]
  | o- pscsi ............................................ [Storage Objects: 0]
  | o- ramdisk .......................................... [Storage Objects: 0]
  | o- user:fbo ......................................... [Storage Objects: 0]
  | o- user:file ........................................ [Storage Objects: 0]
  | o- user:glfs ........................................ [Storage Objects: 0]
  | o- user:qcow ........................................ [Storage Objects: 0]
  | o- user:rbd ......................................... [Storage Objects: 0]
  o- iscsi ....................................................... [Targets: 0]
  o- loopback .................................................... [Targets: 0]
  o- vhost ....................................................... [Targets: 0]
```

Considering everything, we discovered that user space interconnection is not only available for the LIO driver with `target_core_user` module, but it can be simplified via tcmu-runner daemon.

3.3.9. Customizing scripts for device creation

Scripts written for file and block devices were easily customized to create tcmu-runner devices of `user` type: user_create.sh. It requires 3 arguments: handler type (glfs, rbd and etc., but in our light-weight daemon we only support file handler), device name and device size. Script can be found in *6.5. Annex 5: user_create script.*

3.3.10. Creating the configuration with user space device

We have simplified tcmu-runner and removed almost all handlers except file handler.

Let's create a file device:

```
> ./user_create.sh file test_1 4096
 [INFO] Your backstore directory: user_1/test_1
```

All created configuration can be seen using targetcli

```
      o- / ..................................................... [...]
        o- backstores ......................................... [...]
        | o- block ............................... [Storage Objects: 0]
        | o- fileio .............................. [Storage Objects: 0]
        | o- pscsi ............................... [Storage Objects: 0]
        | o- ramdisk ............................. [Storage Objects: 0]
        | o- user:file ........................... [Storage Objects: 1]
        |   o- test_1 ..................... [test_1 (4.0KiB) deactivated]
        o- iscsi ....................................... [Targets: 0]
        o- loopback .................................... [Targets: 0]
        o- vhost ....................................... [Targets: 0]
```

Create target using script:

```
> ./target_create.sh
  [INFO] Target iqn.2017-07.com.test:target-14-17-18 created
  [INFO] Portal: 127.0.0.1:3260
```

Connect target to backstores using script. Specify target iqn and backstore path you got earlier:

```
> ./target_setup.sh iqn.2017-07.com.test:target-14-17-18 user_1/test_1
```

Result can be seen via targetcli:

```
o- backstores ..................................................... [...]
| o- block ......................................... [Storage Objects: 0]
| o- fileio ........................................ [Storage Objects: 0]
| o- pscsi ......................................... [Storage Objects: 0]
| o- ramdisk ....................................... [Storage Objects: 0]
| o- user:file ..................................... [Storage Objects: 1]
|   o- test_1 ............................. [test_1 (4.0KiB) activated]
o- iscsi ................................................... [Targets: 1]
| o- iqn.2017-07.com.test:target-14-17-18 ................... [TPGs: 1]
|   o- tpg1 ................................... [no-gen-acls, no-auth]
|     o- acls ............................................... [ACLs: 1]
|     | o- iqn.1993-08.org.debian:01:ef2e26bf3a9e .... [Mapped LUNs: 1]
|     |   o- mapped_lun1 ..................... [lun1 user/test_1 (rw)]
|     o- luns ............................................... [LUNs: 1]
|     | o- lun1 ........................................ [user/test_1]
|     o- portals ......................................... [Portals: 1]
|       o- 0.0.0.0:3260 ....................................... [OK]
o- loopback ................................................ [Targets: 0]
o- vhost ................................................... [Targets: 0]
```

Connect to target:

```
> iscsiadm -m discovery -t sendtargets -p 127.0.0.1 --login
```

Now we can see devices (`lsscsi`):

```
[0:0:0:0]    disk    VMware,  VMware Virtual S 1.0   /dev/sda
[2:0:0:0]    cd/dvd  NECVMWar VMware IDE CDR10 1.00  /dev/sr0
[3:0:0:1]    disk    LIO-ORG  TCMU device      0002  /dev/sdc
```

Now we can process `dd` to (and from) this device. All data will be stored in a special file created by handler in tcmu-runner folder.

Logout:

```
> iscsiadm -m session -u
```

3.3.11. An overview of tcmu-runner handlers

3.3.12. Writing a customized handler

3.3.13. Creating the configuration with customized handler

3.4. The optimization of capacity

4. Outcomes

5. Literature

1. http://linux-iscsi.org/wiki/LIO
2. https://en.wikipedia.org/wiki/LIO_(SCSI_target)
3. https://en.wikipedia.org/wiki/SCST
4. https://www.rootusers.com/how-to-configure-an-iscsi-target-and-initiator-in-linux/
5. https://www.kernel.org/doc/Documentation/target/tcmu-design.txt
6. https://github.com/open-iscsi/tcmu-runner

6. Annexes

6.1. Annex 1: target_create script

target_create.sh

```
#!/bin/sh

# Script for target creation
# Generates IQN from current date and time
# Creates standard portal 127.0.0.1:3260
# Creates ACL for loopback configuration
# Turns off authorization

# No arguments required

# run as superuser

if [ $(id -u) -ne 0 ]
     then echo "[ERROR] Permission denied. Please run as superuser."
     exit
fi

# check that target driver is loaded

if ! (lsmod | grep -q target); then
     echo "[ERROR] Target driver is not loaded"
     exit
fi

# create iqn

cd /sys/kernel/config/target/iscsi
iqn_name=iqn.$(date +%Y-%m).com.test:target-$(date +%d-%H-%M)
mkdir $iqn_name
cd $iqn_name

# create target portal group

mkdir tpgt_1
cd tpgt_1

# enable target

echo 1 > enable

# create ACL for loopback configuration

cd acls
init_name=`grep '^InitiatorName=' /etc/iscsi/initiatorname.iscsi | sed
's/\(.*\)InitiatorName=\s*\(.*\)/\1\2/'`
mkdir $init_name
cd ..

# set auth params to None
```

```
cd attrib
echo 0 > authentication
cd ..

cd param
echo None > AuthMethod
cd ..

# create portal

cd np
mkdir 0.0.0.0:3260

# info
echo "[INFO] Target $iqn_name created"
echo "[INFO] Portal: 127.0.0.1:3260"
```

6.2. Annex 2: file_create script

file_create.sh

```
#!/bin/sh

# run as root

if [ $(id -u) -ne 0 ]
      then echo "Please run as root"
      exit
fi

# check arguments

if [ $# -ne 3 ]
      then echo "Wrong number of arguments. Must be 3: dir/filename, fileio
dev name, size"
      exit
fi

# create file itself

touch $1
truncate -s $3 $1

# set up backstore for target

cd /sys/kernel/config/target/core

# generate next fileio folder

dir_name=fileio_
idx=0

# If file already exists, find next dir_name that is not yet taken
while true; do
    temp_name=$dir_name$idx
    if [ ! -d $temp_name ]; then
```

```
        mkdir $temp_name
        break
    fi
    idx=$(($idx + 1))
done

# create device

cd $temp_name
mkdir $2
cd $2

echo "Your backstore path: $temp_name/$2"

# configure device

echo "fd_dev_name=$1,fd_dev_size=$3" > control
echo 1 > enable
```

6.3. Annex 3: block_create script

block_create.sh

```
#!/bin/sh


# run as su
if [ $(id -u) -ne 0 ]
     then echo "Please run as root"
     exit
fi


# check args
if [ $# -ne 2 ]
     then echo "Wrong number of arguments. Must be 2: dir/block_file,
iblock dev name"
     exit
fi


# create lvm (lvm2 should be installed)
truncate --size 10M $1
dev_path=`losetup --find --show $1`


cd /dev


vg_name=vol_group_
idx=0
```

```
# If file already exists, find next dir_name that is not yet taken
while true; do
    temp_name=$vg_name$idx
    if [ ! -d $temp_name ]; then
        break
    fi
    idx=$(($idx + 1))
done
vg_name=$temp_name
vgcreate $vg_name $dev_path
lvcreate --size 4M --name log_vol $vg_name

# create backstore for target
cd /sys/kernel/config/target/core

# generate next iblock folder
dir_name=iblock_
idx=0

# If file already exists, find next dir_name that is not yet taken
while true; do
    temp_name=$dir_name$idx
    if [ ! -d $temp_name ]; then
        mkdir $temp_name
        break
    fi
    idx=$(($idx + 1))
done

# create device
cd $temp_name
mkdir $2 # dev name
cd $2

echo "Your backstore path: $temp_name/$2"
```

```
# configure device
echo "udev_path=/dev/$vg_name/log_vol" > control
echo 1 > enable
```

6.4. Annex 4: target_setup script

target_setup.sh

```
#!/bin/sh

# Script that sets up device to existing target
# Creates lun

# Input arguments required:
## 1. Target IQN
## 2. Backstore path (got from script)
# Example: ./target_setup.sh iqn.2017-07.com.example:target file_0/test

# run as superuser

if [ $(id -u) -ne 0 ]
     then echo "[ERROR] Permission denied. Please run as superuser."
     exit
fi

# check arguments

if [ $# -ne 2 ]
then echo "[ERROR]         Wrong number of arguments. Must be 2."
     echo "[ARGUMENTS]     target IQN, backstore path."
     exit
fi

cd /sys/kernel/config/target/iscsi/$1/tpgt_1

# generate next lun

cd lun
dir_name=lun_
idx=0

# if folder already exists, find next name that is not yet taken

while true; do
    lun_name=$dir_name$idx
    if [ ! -d $lun_name ]; then
        mkdir $lun_name
        break
    fi
    idx=$(($idx + 1))
done

cd $lun_name

ln -s /sys/kernel/config/target/core/$2
```

```
cd ../..

# create acls (only for loopback configuration)

cd acls
init_name=`grep '^InitiatorName=' /etc/iscsi/initiatorname.iscsi | sed
's/\(.*\)InitiatorName=\s*\(.*\)/\1\2/'`
cd $init_name

mkdir $lun_name
cd $lun_name
ln -s /sys/kernel/config/target/iscsi/$1/tpgt_1/lun/$lun_name/
```

6.5. Annex 5: user_create script

user_create.sh

```
#!/bin/sh


# Script for user-backstores creation
# Creates entries in sysfs for backstore device


# Input arguments required:
## 1. Handler type ('file' in our light-weight tcmu-runner)
## 2. Device name (must be unique for one target)
## 3. Device (file or memory) size
# Example: ./user_create.sh alloc test_dev 1048576


# run as superuser
if [ $(id -u) -ne 0 ]
     then echo "[ERROR] Permission denied. Please run as superuser."
     exit
fi


# check arguments
if [ $# -ne 3 ]
then echo "[ERROR]         Wrong number of arguments. Must be 3."
      echo "[ARGUMENTS]    handler type, device name, device size."
      echo "[EXAMPLE]./user_create.sh alloc test_dev 1048576"
     exit
fi


# check that target_core_user module is loaded
if ! (lsmod | grep -q target_core_user); then
```

```
      echo "[ERROR] target_core_user module is not loaded"
      exit
fi

# create backstore object in sysfs
cd /sys/kernel/config/target/core

# generate next user backstore
dir_name=user_
idx=0

# if folder already exists, find next name that is not yet taken
while true; do
    temp_name=$dir_name$idx
    if [ ! -d $temp_name ]; then
        mkdir $temp_name
        break
    fi
    idx=$(($idx + 1))
done

cd $temp_name
mkdir $2
cd $2

# configuration
echo -n dev_size=$3 > control
echo -n dev_config=$1/$2 > control
echo -n 1 > enable

# use this info to setup target for device using target_setup script
echo "[INFO] Your backstore directory: $temp_name/$2"
```