Project report

**Replacement of SCST iSCSI driver with in-kernel Linux-IO driver**

Project manager Yuri Stotski

Project executors Denis Koptev, Vladislav Kulakovsky

Contents

[Introduction 3](#_Toc494330279)

[Basic goal 3](#_Toc494330280)

[Tasks of the project 3](#_Toc494330281)

[Works done 4](#_Toc494330282)

[Overview of Linux-IO driver 4](#_Toc494330283)

[Backstores 4](#_Toc494330284)

[iSCSI 5](#_Toc494330285)

[Targetcli 5](#_Toc494330286)

[Comparison with SCST driver 5](#_Toc494330287)

[Possibility of replacement of SCST driver with Linux-IO driver 6](#_Toc494330288)

[Creation of standard LIO configuration via targetcli interface 6](#_Toc494330289)

[Setting up the initiator side 8](#_Toc494330290)

[Review of the configuration in sysfs 9](#_Toc494330291)

# 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.

## 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.

## 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.

# Works done

## 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.

### Backstores

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.

As a result, LIO provides a generalized model to export block storage.

### 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.

### 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.

## 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.

## 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.

### Creation of 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.

### 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.

### 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.