DOCA Storage Zero Copy Target RDMA Application Guide

Table of contents

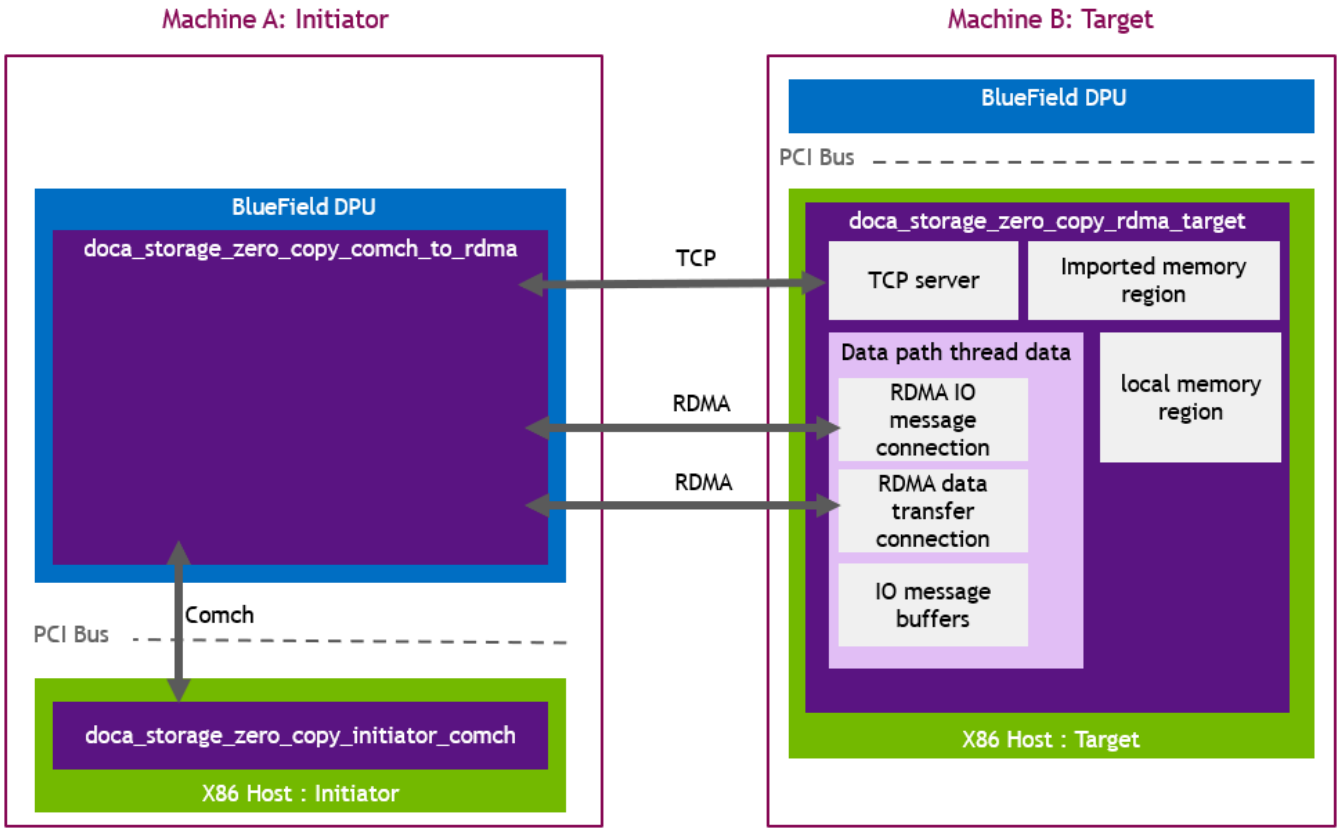
Introduction

DOCA Storage Zero Copy Target RDMA (target_rdma) acts as a mock storage service, preparing an area of memory equal in size to the block created by the `doca_storage_zero_copy_initiator_comch` (initiator_comch). This application waits for IO messages from the `doca_storage_zero_copy_comch_to_rdma` (comch_to_rdma) and performs the necessary RDMA read or write operations to fulfill the initiators' read or write request (i.e., RDMA write for a read IO message, DMA read for a write IO message).

System Design

DOCA Storage Zero Copy Target RDMA uses a TCP socket for out-of-band control messages, then uses two DOCA RDMA connections:

- One for the data path to receive and reply to IO messages; and
- Another to perform the RDMA read and write operations which actually move data to or from the memory created by initiator_comch



Application Architecture

DOCA Storage Zero Copy Target RDMA executes in three stages:

1. Preparation.
2. Data path.
3. Teardown.

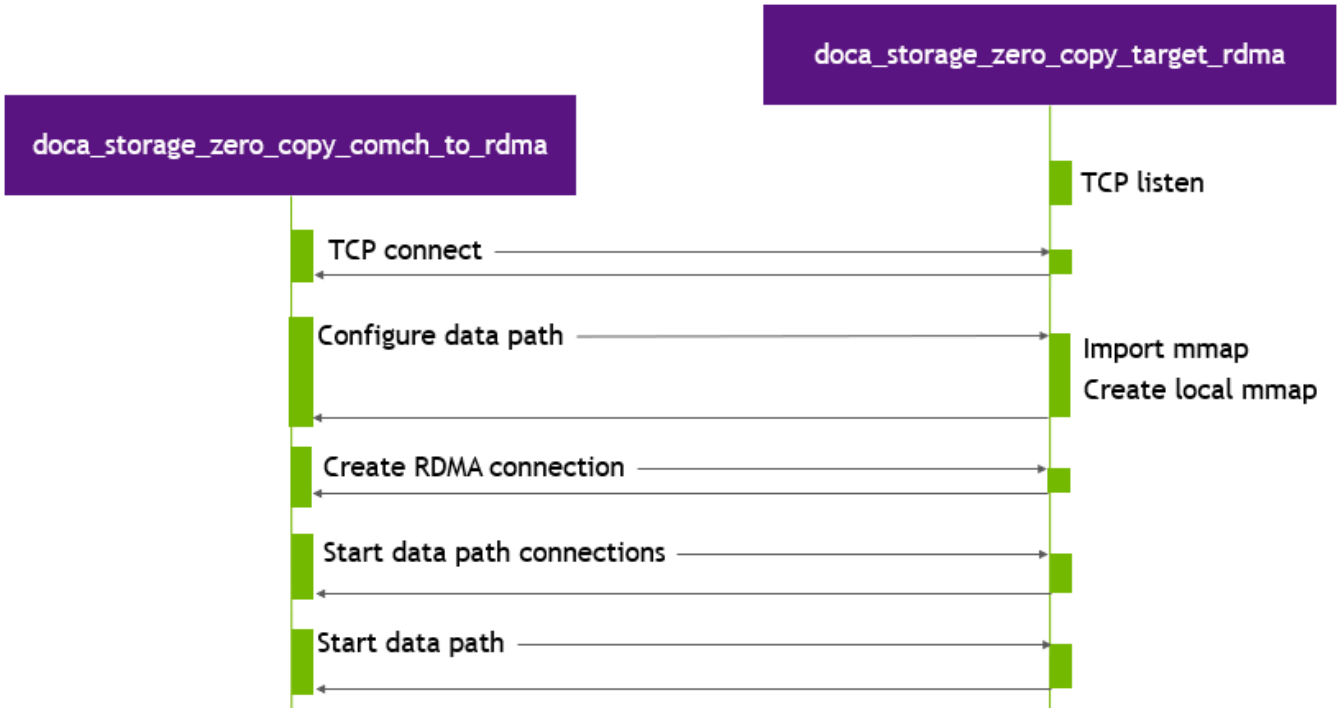
Preparation Stage

During this stage the application performs the following:

1. Creates a TCP server socket.
2. Waits for comch_to_rdma to connect.
3. Waits for a configure data path control message (buffer count, buffer size, doca_mmap export details) from comch_to_rdma.
    1. Imports the received doca_mmap.
    2. Create a local memory region.
    3. Creates a local doca_mmap.
    4. Creates a doca_buf_inventory.
    5. Sends a configure data path control message response to comch_to_rdma.
4. Waits for `N` "create RDMA connection" control messages from comch_to_rdma.
    1. Creates the RDMA context.
    2. Exports the connection details.
    3. Starts connecting using the provided remote connection details.
    4. Sends a create RDMA connection control message response to comch_to_rdma.
5. Waits for a "start data path connections" control message from comch_to_rdma.

1. Verifies that all RDMA connections are ready to use.
2. Sends a start data path connections control message response to comch_to_rdma.

6. Waits for a start storage control message from comch_to_rdma.
    1. Starts data path threads.
    2. Sends a start storage control message response to comch_to_rdma.



Data Path Stage

In this stage, the data path threads start. Each thread begins by submitting receive RDMA tasks then executing a tight loop and polling the progress engine (PE) as quickly as possible until a "data path stop" IO message is received.

The process of handling an IO message involves the following steps:

1. Determine memory locations to be used for decoding the IO message.
2. Submit a RDMA read/RDMA write operation.
3. Upon completion of the RDMA read/write, send a response IO message to BlueField.

4. Resubmit the RDMA receive task.

Teardown Stage

In this stage the application performs the following:

1. Waits for a destroy objects control message from.
2. Destroys data path objects.
3. Sends a destroy objects control message response to comch_to_rdma.
4. Destroys control path objects.

DOCA Libraries

This application leverages the following DOCA libraries:

- DOCA RDMA

Compiling the Application

This application is compiled as part of the set of storage zero copy applications. For compilation instructions, refer to *NVIDIA DOCA Storage Zero Copy*.

Running the Application

Application Execution

DOCA Storage Zero Copy Comch to RDMA is provided in source form. Therefore, a compilation is required before the application can be executed.

- Application usage instructions:

```
Usage: doca_storage_zero_copy_target_rdma [DOCA Flags]
[Program Flags]
```

```
DOCA Flags:
  -h, --help                            Print a help synopsis
  -v, --version                         Print program version
information
  -l, --log-level                       Set the (numeric) log
level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR,
40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  --sdk-log-level                       Set the SDK (numeric) log
level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR,
40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  -j, --json <path>                     Parse all command flags
from an input json file

Program Flags:
  -d, --device                          Device identifier
  -r, --representor                     Device host side
representor identifier
  --listen-port                         TCP Port on which to
listen for incoming connections
  --cpu                                 CPU core to which the
process affinity can be set
```

Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./ doca_storage_zero_copy_target_rdma -h
```

For additional information, refer to section "Command Line Flags".

- CLI example for running the application on the BlueField:

```
./doca_storage_zero_copy_target_rdma -d 03:00.0 --listen-port 12345 --cpu 12
```

> **Info**
>
> The DOCA device PCIe address, `3b:00.0`, should match the address of the desired PCIe device.

- The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_storage_zero_copy_target_rdma --json [json_file]
```

For example:

```
./doca_storage_zero_copy_target_rdma --json doca_storage_zero_copy_comch_to_rdma_params.json
```

> **Note**

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	`h`	`help`	Print a help synopsis	N/A
	`v`	`version`	Print program version information	N/A
	`l`	`log-level`	Set the log level for the application: • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 (requires compilation with `TRACE` log level support)	`"log-level": 60`
	N/A	`sdk-log-level`	Set the log level for the program: • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70	`"sdk-log-level": 40`

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	`j`	`json`	Parse all command flags from an input JSON file	N/A
Program flags	`d`	`device`	DOCA device identifier. One of: • PCIe address – `3b:00.0` • InfiniBand name – `mlx5_0` • Network interface name – `en3f0pf0sf0` **Note** This is a mandatory flag.	`"device": "03:00.0"`
	N/A	`--listen-port`	TCP port on which to listen for incoming connections **Note** This is a mandatory flag.	`"lister-port": 12345`
	N/A	`--cpu`	Index of CPU to use. One data path thread is spawned per CPU. Index starts at 0. **Note** The user can	`"cpu": 6`

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			specify this argument multiple times to create more threads. **Note** This is a mandatory flag.	

Troubleshooting

Refer to the DOCA Troubleshooting for any issue encountered with the installation or execution of the DOCA applications.

Application Code Flow

Control Thread Flow

1. Parse application arguments:

```
auto const cfg = parse_cli_args(argc, argv);
```

1. Prepare the parser (`doca_argp_init`).
2. Register parameters (`doca_argp_param_create`).
3. Parse the arguments (`doca_argp_start`).
4. Destroy the parser (`doca_argp_destroy`).

2. Display the configuration:

```
print_config(cfg);
```

3. Create application instance:

```
g_app.reset(storage::zero_copy::make_storage_application(cfg))
```

4. Run the application:

```
g_app->run()
```

1. Find and open the specified device:

```
m_dev = storage::common::open_device(m_cfg.device_id);
```

2. Start the TCP server and wait for comch_to_rdma to connect:

```
start_listening();
wait_for_tcp_client();
```

3. Wait for a "configure storage" control message from comch_to_rdma.
4. Configure storage:

```
configure_storage(configuration);
```

    1. Create thread contexts:
        1. Create transaction contexts.
        2. Create IO messages.
        3. Create PE.
        4. Create mmap for IO message buffers.
5. Send configure storage control message response to comch_to_rdma.
6. Wait for `N` "create RDMA connection" control messages from comch_to_rdma:
    1. Create RDMA context.
    2. Export connection details.
    3. Start connection using received remote connection details.
    4. Send a "create RDMA connection" control message response (containing RDMA connection details from target_rdma RDMA context) to comch_to_rdma.
7. Wait for "start data path" control message from comch_to_rdma:
    1. Verify all connections are ready (comch and RDMA):

```
establish_rdma_connections();
```

8. Send a "start storage" control message response to comch_to_rdma.
9. Wait for start storage control message from comch_to_rdma:

1. Create data path threads.
2. Start data path threads.

10. Send a "start storage" control message response to comch_to_rdma.
11. Run all threads until completion.
12. Wait for "destroy objects" control message.
13. Destroy data path objects.
14. Send destroy objects control message response to BlueField.

5. Display stats:

```
printf("+=============================================+\n");
printf("| Stats\n");
printf("+=============================================+\n");
for (uint32_t ii = 0; ii != stats.size(); ++ii) {
        printf("| Thread[%u]\n", ii);
        auto const pe_hit_rate_pct = (static_cast<double>
(stats[ii].pe_hit_count) /
                                        (static_cast<double>
(stats[ii].pe_hit_count) +
                                         static_cast<double>
(stats[ii].pe_miss_count))) *
                                    100.;
        printf("| PE hit rate: %2.03lf%% (%lu:%lu)\n",
                pe_hit_rate_pct,
                stats[ii].pe_hit_count,
                stats[ii].pe_miss_count);

        printf("+---------------------------------------------+\n");
}
printf("+=============================================+\n");
```

6. Destroy control path objects.

Performance Data Path Thread Flow

The data path involves polling the PE as quickly as possible to receive IO messages from BlueField.

1. Run until BlueField sends a stop IO message:

```
while (hot_data->running_flag) {
        doca_pe_progress(pe) ? ++(hot_data->pe_hit_count) :
++(hot_data->pe_miss_count);
}
```

2. Handle BlueField IO message:

    1. Calculate memory addresses to use for local and remote memory.

    2. Set buffer addresses and sizes into source and destination buffers into RDMA task.

    3. Start RDMA read/write task.

    4. Upon completion of RDMA task respond to BlueField.

    5. Re-submit RDMA recv task.

References

- `/opt/mellanox/doca/applications/storage/`

and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.
