DOCA Storage Zero Copy Comch to RDMA Application Guide

Table of contents

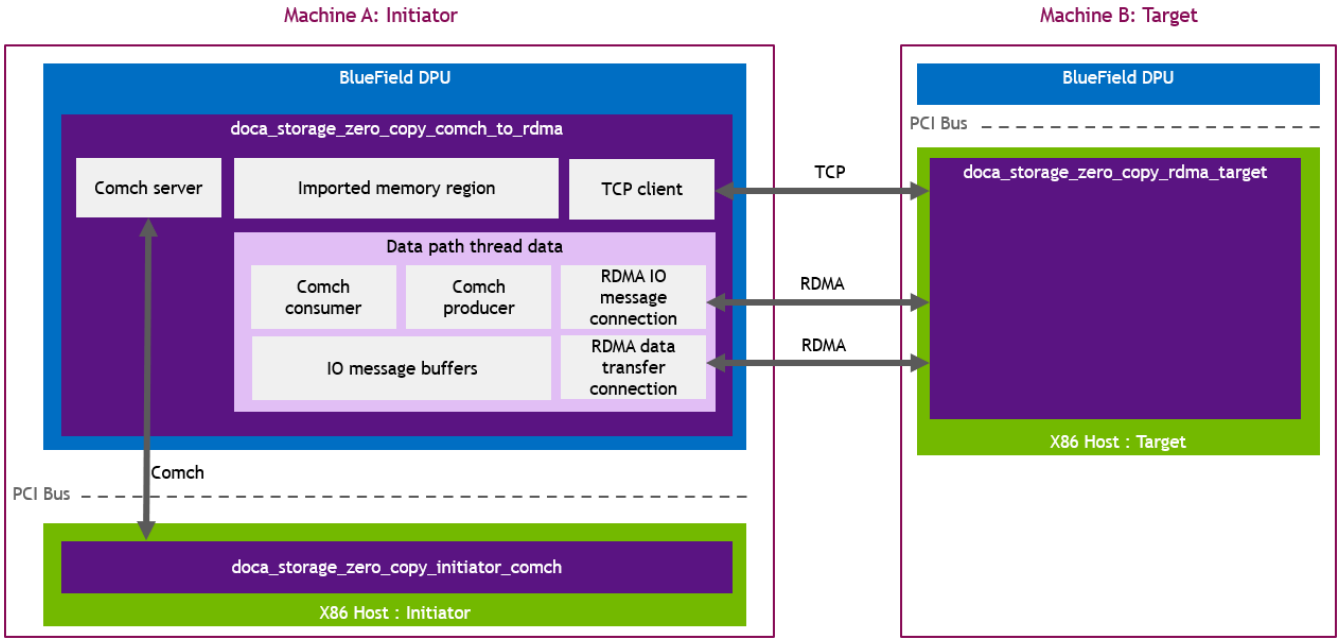
Introduction

DOCA Storage Zero Copy Comch to RDMA (comch_to_rdma) is a communications bridge between the `doca_storage_zero_copy_initiator_comch` (initiator_comch) and the `doca_storage_zero_copy_target_rdma` (target_rdma). This keeps the initiator_comch insulated from the details of target_rdma.

System Design

1. Comch_to_rdma connects to target_rdma via TCP.
2. Comch_to_rdma creates a comch server and waits for the initiator_comch to connect.
3. Comch_to_rdma waits for control messages from the initiator_comch and reacts to them appropriately.

> **Info**
>
> Two RDMA connections are made per thread to avoid the large RDMA data transfers interfering with or introducing latency to the smaller IO messages.



Application Architecture

DOCA Storage Zero Copy Comch to RDMA executes in three stages:

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

Preparation Stage

During this stage, the application performs the following:

1. Connects to target_rdma via TCP.
2. Creates a DOCA Comch server and waits for a client connection.
3. Waits for a "configure data path" control message from initiator_comch (including buffer count, buffer size, doca_mmap export details).
    1. Create a doca_mmap using the exported details from initiator_comch then re-export it to provide access to target_rdma.
    2. Send a configure data path control message to target_rdma.

3. Wait for a configure data path control message response with a success status from target_rdma.

4. Send a configure data path control message response to initiator_comch.

4. Waits for a "start data path connections" control message from initiator_comch.

    1. Create comch data path objects.

    2. Create `N` RDMA connections, exchanging connection details with target_rdma.

    3. Relay the start data path connections control message to target_rdma.

    4. Wait for a start data path connections control message response with a success status from target_rdma.

    5. Send a start data path connections control message response to initiator_comch.

5. Waits for a "start storage" control message from initiator_comch.

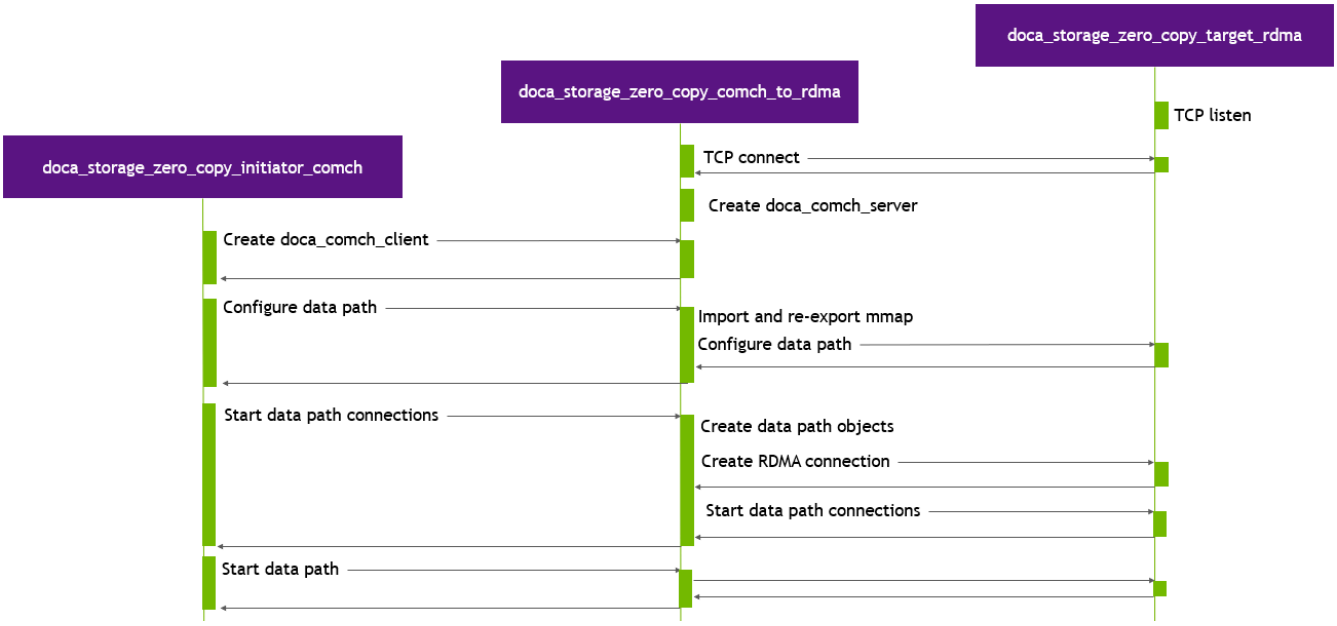
    1. Verify that all RDMA and Comch connections are ready to use.

    2. Send a start storage control message to target_rdma.

    3. Wait for a start storage control message response with a success status from target_rdma.

    4. Start data path threads.

    5. Send a start storage control message response to initiator_comch.



Data Path Stage

This stage starts the data path threads. Each thread begins by submitting receive comch and RDMA tasks, then executing a tight loop polling the progress engine (PE) as quickly as possible until a "data path stop" IO message is received. The work of the data path threads is reactive, so is performed in task completion callbacks. As each IO message is received from initiator_comch, it is forwarded to the storage application. Similarily, as each IO message response is received from target_rdma, it is relayed back to initiator_comch.

Teardown Stage

In this stage, the application performs the following:

1. Wait for a destroy objects control message from initiator_comch.
2. Send a destroy objects control message to target_rdma.
3. Wait for a destroy objects control message response from target_rdma.
4. Destroy data path objects.
5. Send a destroy objects control message response to initiator_comch.

6. Destroy control path objects.

DOCA Libraries

This application leverages the following DOCA libraries:

- DOCA Comch
- 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

> **Note**
>
> This application can only be run on the host.

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

- Application usage instructions:

```
Usage: doca_storage_zero_copy_comch_to_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
  --cpu                            CPU core to which the
process affinity can be set
  --storage-server                 One or more storage
server addresses in <ip_addr>:<port> format
  --command-channel-name           Name of the channel used
by the doca_comch_server. Default: storage_zero_copy_comch
```

Info

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

```
./doca_storage_zero_copy_comch_to_rdma -h
```

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

- CLI example for running the application on the BlueField:

```
./doca_storage_zero_copy_comch_to_rdma -d 03:00.0 -r 3b:00.0
--storage-server 172.17.0.1:12345 --cpu 12
```

> **Note**
>
> Both the DOCA Comch device PCIe address (`03:00.0`) and the DOCA Comch device representor PCIe address (`3b:00.0`) should match the addresses of the desired PCIe devices.

- 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_comch_to_rdma --json [json_file]
```

For example:

```
./doca_storage_zero_copy_comch_to_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`
	`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`	`"device" : "03:00.0"`

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			Note This flag is a mandatory.	
	r	representor	DOCA Comch device representor PCIe address **Note** This flag is a mandatory.	"representor" : "3b:00.0"
	N/A	--cpu	Index of CPU to use. One data path thread is spawned per CPU. Index starts at 0. **Note** The user can specify this argument multiple times to create more threads. **Note** This flag is a mandatory.	"cpu" : 6
	N/A	--storage-server	IP Address and port to use to establish the control TCP connection to the target. **Note**	"storage-server" :

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
			This flag is a mandatory.	"172.17.0.1:12345"
	N/A	`--command-channel-name`	Allows customizing the server name used for this application instance if multiple comch servers exist on the same device.	"command-channel-name" : "storage_zero_copy_comch"

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_dpu_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. Find and open the selected representor:

```
m_dev_rep = storage::common::open_representor(m_dev,
m_cfg.representor_id);
```

3. Create control path progress engine:

```
doca_pe_create(&m_ctrl_pe);
```

4. Connect to target_rdma:

```
connect_storage_server();
```

    1. Create a TCP socket.
    2. Connect the TCP socket.

5. Create comch server and wait for comch client to connect:

```
create_comch_server();

while (m_client_connection == nullptr) {
        static_cast<void>(doca_pe_progress(m_ctrl_pe));

        if (m_abort_flag)
                return;
}
```

6. Wait for configure storage control message.
7. Configure storage:

```
configure_storage();
```

    1. Create mmap using the exported details provided by initiator_comch.
    2. Export the mmap to allow RDMA access.

8. Send "configure storage" control message to target_rdma with re-exported mmap details.
9. Wait for configure storage control message response from target_rdma.

10. Send configure storage control message response to initiator_comch.
11. Wait for "start data path" control message.
12. Prepare data path:

```
for (uint32_t ii = 0; ii != m_cfg.cpu_set.size(); ++ii) {
        prepare_storage_context(ii, msg.correlation_id);
}
```

    1. Create per thread data context:
        1. Create IO messages.
        2. Create progress engine.
        3. Create mmap for IO message buffers.
        4. Create comch producer.
        5. Create comch consumer.
        6. Create RDMA contexts.
        7. Create RDMA connections:
            1. Export RDMA connection details (`doca_rdma_export`).
            2. Send "create RDMA connection" control message.
            3. Wait for create RDMA connection control message.
            4. Start connection using remote RDMA connection details `doca_rdma_connect`.
    2. Send data path control message to target_rdma.
    3. Wait for data path control message response from target_rdma.
    4. Send data path control message response to initiator_comch.

13. Wait for start storage control message.

14. Verify all connections are ready (comch and RDMA):

```
wait_for_connections_to_establish();
```

15. Send start storage control message to target_rdma.

16. Create threads:

```
if (op_type == io_message_type::read) {
        m_thread_contexts[ii].thread =
std::thread{&thread_hot_data::non_validated_test,

std::addressof(m_thread_contexts[ii].hot_context)};
} else if (op_type == io_message_type::write) {
        if (m_cfg.validate_writes) {
                m_thread_contexts[ii].thread =

std::thread{&thread_hot_data::validated_test,

std::addressof(m_thread_contexts[ii].hot_context)};
        } else {
                m_thread_contexts[ii].thread =

std::thread{&thread_hot_data::non_validated_test,

std::addressof(m_thread_contexts[ii].hot_context)};
        }
}
```

17. Wait for "start storage" control message response from target_rdma.

18. Start data path threads.

19. Send start storage control message response to initiator_comch.
20. Run all threads until completion.
21. Wait for "destroy objects" control message.
22. Send destroy objects control message to target_rdma.
23. Wait for destroy objects control message response from target_rdma.
24. Destroy data path objects.
25. Send destroy objects control message response to initiator_comch.

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 either initiator_comch or target_rdma.

1. Run until initiator_comch 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 IO message from initiator_comch:

```
auto *const hot_data = static_cast<thread_hot_data *>
(ctx_user_data.ptr);
...
doca_task_submit(static_cast<doca_task *>(task_user_data.ptr));
```

3. Handle IO message from target_rdma:

```
auto *const hot_data = static_cast<thread_hot_data *>
(ctx_user_data.ptr);
doca_error_t ret;

auto *const io_message =
storage::common::get_buffer_bytes(doca_rdma_task_receive_get_d
```

```
if (io_message_view::get_type(io_message) !=
io_message_type::stop) {
        io_message_view::set_type(io_message_type::result,
io_message);
        io_message_view::set_result(DOCA_SUCCESS,
io_message);
} else {
        hot_data->app_impl->stop_all_threads();
}

do {
        ret = doca_task_submit(static_cast<doca_task *>
(task_user_data.ptr));
} while (ret == DOCA_ERROR_AGAIN);
```

References

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

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.
