Journal of Physics: Conference Series **608** (2015) 012029 doi:10.1088/1742-6596/608/1/012029

High-speed zero-copy data transfer for DAQ applications

Flavio Pisani [1,2], Daniel Hugo Cámpora Pérez[1] and Niko Neufeld[1]

[1] CERN, CH-1211 Geneva 23, CH

[2] Dipartimento di fisica, Università degli Studi di Roma "La Sapienza", Piazzale Aldo Moro 5, 00185 Roma, IT

E-mail: flavio.pisani@cern.ch

Abstract. The LHCb Data Acquisition (DAQ) will be upgraded in 2020 to a trigger-free readout. In order to achieve this goal we will need to connect around 500 nodes with a total network capacity of 32 Tb/s. To get such an high network capacity we are testing zero-copy technology in order to maximize the theoretical link throughput without adding excessive CPU and memory bandwidth overhead, leaving free resources for data processing resulting in less power, space and money used for the same result. We develop a modular test application which can be used with different transport layers. For the zero-copy implementation we choose the OFED IBVerbs API because it can provide low level access and high throughput. We present throughput and CPU usage measurements of 40 GbE solutions using Remote Direct Memory Access (RDMA), for several network configurations to test the scalability of the system.

1. Introduction

The LHCb [1] experiment is one of the four big experiments placed in the Large Hadron Collider (LHC), situated in Geneva Switzerland. The detector collects data from proton-proton collisions at design center of mass energy of 14 TeV and a design rate of 40 MHz. To handle the big amounts of data produced by the detector the Data Acquisition System (DAQ) selects interesting events according to the underling physics in a multi-stage process usually called trigger.

In 2019-2020 the LHCb DAQ system will be upgraded in order to sustain the increasing amounts of data produced by the detector and to filter the data without a low level hardware trigger. Figure 1 shows the network design of the system, which follows a two-stage design. Readout Units (RUs) gather subdetector-specific incoming data fragments, and distribute them across Builder Units (BUs), following a many-to-one network flow pattern. The receiving BU is selected by a load-balanced distribution, managed by a centralized dispatcher named Event Manager (EM), and data is sent over a 100 Gb/s based event building network. In the next stage, the Filtering Units (FUs) receive the data from the BUs in a one to one pattern via a 10 Gb/s network. A more detailed description of the LHCb DAQ network upgrade can be found in [2].

We present a zero-copy implementation of a transport layer which is handling the event building traffic between RUs, BUs and EM. This transport layer is integrated in the Data Acquisition Protocol Independent Evaluator [2] (DAQPIPE) emulation software, which emulates an DAQ network traffic and makes possible to measure the performances of different network technologies.

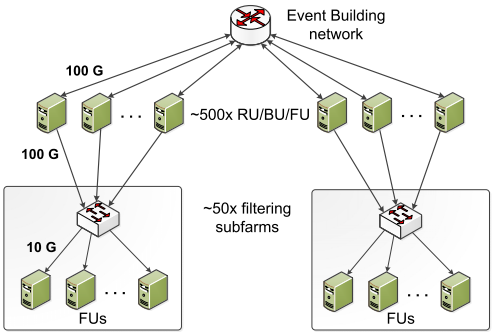

Figure 1. The LHCb upgrade network design.

2. Remote Direct Memory Access (RDMA)

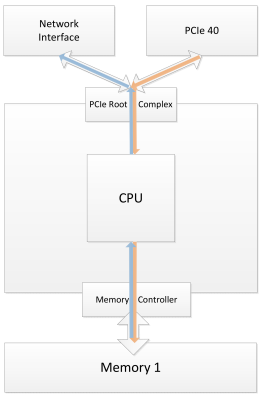

Figure 2. Example of non-RDMA data transfer from a PCIe read-out card to a network interface.

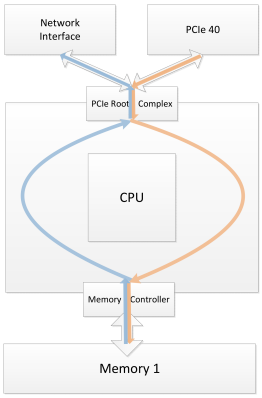

Figure 3. Example of RDMA data transfer from a PCIe read-out card to a network interface.

Traditional data transfer over the network requires, as shown in Figure 2, several operations to be performed by the CPU and the OS: moving the data between application space and kernel space; handling all the steps required by the protocol stack currently in use; moving the data between the memory and the network card. These operations produce CPU time and memory bandwidth overhead which is increasing with the increasing bandwidth that can be achieved with modern network technologies, for example 40/100 GbE or Infiniband [3].

Using RDMA data transfers, the data are written to or read from addresses in application space directly by the network card, as shown in Figure 3. For transferring data in this schema the user has to provide an available physical memory address space on both the local and the remote machine. This process, referred to as memory registration, is CPU intensive and locks the physical addresses on the RAM, on the other hand using RDMA removes all CPU and memory overhead introduced during the transmission itself. The address of the remote node needs to be exchanged over the network adding a small transmission overhead every time we want to transfer a message to a new destination memory address. Due to the nature of the overheads introduced by an RDMA transfer it is a suitable solution for transferring big chunks of data and reusing memory buffers where possible.

3. Protocol and API

In this section we briefly expose the essential parts of the IBVerbs API [4] and the protocol used in our DAQ implementation.

3.1. The IBVerbs API

The IBVerbs library offers a user space API to access the RDMA capabilities of different devices across a variety of network technologies. The library allows the final user to abstract from the complexity of the hardware providing an hardware independent structure.

Below we discuss OFED-specific terminology [5] and aspects of the IBVerbs library which are necessary to understand the problems that may occur in a DAQ application and how to solve them. Further informations can be found in [6] and [7].

- **Work queue (WQ):** FIFO (First In First Out) queue used by the user to issue a work request to the network card. The work requests are then executed in a non-blocking way.
- **Completion queue (CQ):** FIFO queue used by the network card to notify the completion of a previously posted work request. Several WQs may share the same CQ.
- **Memory region (MR):** Chunk of memory ready to be used as source or destination of a data transfer.
- **Send op-code:** Sends data from a registered MR. There is no need to know the remote memory region but a *receive* request must be posted, on the receiving WQ by the receiver of the message, before the message is sent.
- **Receive op-code:** Writes data to a registered MR. Must be posted to the WQ before the *send* request.
- **RDMA write op-code:** Writes data from a locally registered MR to a remotely registered MR. The remote node does not receive any notification in the CQ when the transfer is over, the sender has to communicate this to the receiver.

Registering a new MR is a very time-consuming process which requires several operations to be performed by the OS and the driver [8]. In order to avoid registration and deregistration of the memory for every transfer, it is preferable to use designated memory buffers, which are registered only once.

The *Send/Receive* operations can be used to send small control messages to preregistered temporary MRs without introducing the heavy overhead of an *RDMA write* operation. To receive a *send* operation properly, a corresponding *receive* must have been posted to the WQ. To achieve this, one possible strategy, is to post enough *receive* request to receive the maximum number of messages that can be sent at once.

On the other hand the use of temporary buffers and the large number of work requests issued at the same time make *Send/Receive* a bad choice for transferring large data chunks to a specific memory location without any additional copy. The *RDMA write* is, at the contrary, a very suitable solution for sending large amount of data. To minimize the overhead introduced by memory registration a good practice, for a DAQ application, it is to define and register a large buffer in memory and to write the data to a subset of it.

3.2. Protocol

Figure 4 depicts all the messages exchanged during the event building process for two different protocols, namely push and pull. The protocol determines the communication starter between RUs and BUs, in the push implementation the transfer is initiated by the RUs, in the pull implementation by the BUs. In both protocols the event is assigned to a specific BU by the EM every time there is a new trigger event. The trigger frequency can be configured at run time.

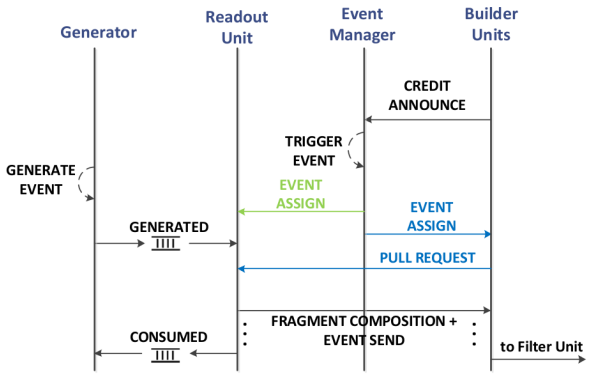

Figure 4. Protocol flow design. Push-specific messages are depicted in light green, and pull-specific ones are depicted in blue.

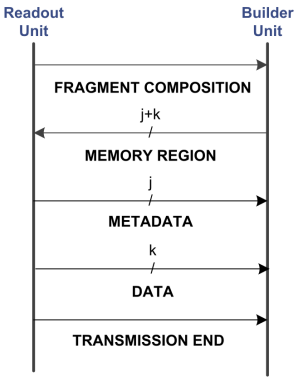

Figure 5. Fragment composition and event send flow design using RDMA transport layer.

The workload is balanced between the BUs using a credit system. The available resources are divided in slots, every time there is a free slot on a BU a *credit announce* message is generated. The EM assigns the events to the BUs according to the available slots. A more detailed description of the protocols and the messages exchanged is described in [2].

Figure 5 shows more in detail the messages exchanged between RUs and BUs in order to send the event fragments using an IBVerbs transport implementation. A full description of the messages follows:

(i) **FRAGMENT COMPOSITION:** The RU sends a fragment composition structure, which contains the number and the sizes of metadata and data fragments that are going to be sent, using a *send* operation.

(ii) **MEMORY REGION:** The BU sends, using a *send* operation, a valid MR for each fragment.

(iii) **METADATA and DATA:** The RU sends the metadata and the data fragments using and *RDMA write* operation.

(iv) **TRANSMISSION END:** The RU sends, using a *send* operation, a transmission complete message including the event ID of the completed event.

The need to send a MR before transferring the data and the metadata makes the protocol inherently pull, a possible way to avoid this is to use preregistered MR sent during the initialization of the system. Because each RU cannot calculate the exact amount of data collected by all the others RUs without gathering all the *fragment compositions*, to avoid dangerous overlaps in the data buffer, the RU must take into account the maximum possible size for every event resulting in higher memory usage.

4. Bandwidth and CPU usage Measurements

In order to test the performances of the transport layer implementation, we have built a test system consistent of 5 nodes with the hardware and software described in table 1. The nodes are connected through a Brocade ICX 7750 40 GbE switch and the network cards are configured for operating in Ethernet mode. In order to reduce the network overhead we enabled jumbo frames[1] on both the network cards and the switch.

[1] Jumbo frames are Ethernet frames with a payload greater that 1500 bytes.

Table 1. Server characteristics

CPU	2 x Intel Xeon CPU E5-2670 @ 2.60 GHz
Memory	32 GB
Network card	Mellanox MT27500 Connectx3
OFED version	Mellanox 2.2-1.0.1
Linux version	slc 6.5 kernel 2.6.32-431.23.3

Table 2 shows the effective event building throughput achieved using both protocols for different number of nodes. In the different configurations tested the effective event building throughput is very close the theoretical maximum and scales up in a consistent way changing the number of nodes in the system for both push and pull.

Table 2. Effective event building throughput for different configurations

Number of nodes	pull throughput [Gb/s]	push throughput [Gb/s]
2	38.3 (95.7%)	38.3 (95.7%)
3	39.0 (97.6%)	39.0 (97.4%)
4	39.0 (97.5%)	39.1 (97.6%)
5	38.8 (96.9%)	38.8 (97.0%)

The total CPU usage, calculated using sar[2], for the DAQPIPE application using the IBVerbs transport is 0.94%. A full duplex transmission done by iperf reports a system CPU usage of 6.8%, we can provide an estimation of 86% lower CPU usage using RDMA transport.

5. Conclusions and future work

The Verbs implementation of the event building transport layer is performing according to the specifications in all the tested scenarios: push and pull protocol connecting up to 5 nodes through the ICX 7750 switch. The CPU usage is 86% lower using RDMA transport, leaving free resources for other tasks, like for opportunistic High Level Trigger (HLT) execution.

The scalability of the system up to 5 nodes is very promising, but larger scale tests will be performed in the short future. The many to one nature of the traffic makes the scalability to hundreds of nodes a challenging task, not only from an application point of view, but also from a networking perspective.

We will continue testing 40 Gb/s technologies looking forward to 100 Gb/s in the future, which should provide enough bandwidth for a complete trigger-free readout.

[2] The sar command writes to standard output the activity counters of the operating system. For more information see the sar manual.

References

[1] The LHCb Collaboration et al 2008 *Journal of Instrumentation* **3** S08005
[2] Cámpora Pérez D H, Schwemmer R and Neufeld N *Protocol-Independent Event Building Evaluator for the LHCb DAQ System* (Proceedings of RT2014. Submitted to Transactions of Nuclear Science, pending review.)
[3] Infiniband Trade Association (IBTA) URL `http://www.infinibandta.org/`
[4] Mellanox Technologies 2001 *Mellanox IB-Verbs API (VAPI)*
[5] Open Fabric Alliance (OFED) URL `https://www.openfabrics.org/index.php`
[6] Grun P 2010 *Introduction to infiniband for end users*
[7] Mellanox Technologies 2014 *RDMA Aware Programming user manual*
[8] Mietke F, Baumgartl R, Rex R, Mehlan T, Hoefler T and Rehm W 2006 *Proceedings of Euro-Par 2006 Parallel Processing* (Springer-Verlag Berlin) pp 124–133 ISBN 3-540-37783-2