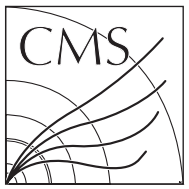
Available on CMS information server

CMS CR -2012/135



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

10 June 2012 (v3, 15 June 2012)

A comprehensive zero-copy architecture for high performance distributed Data Acquisition over advanced network technologies for the CMS experiment

G. Bauer[6)], U. Behrens[1)], J. Branson[4)], S. Bukowiec[2)], O. Chaze[2)],S. Cittolin[4)], J. A. Coarasa[2)], C. Deldicque[2)], M. Dobson[2)], A. Dupont[2)], S. Erhan[3)], D. Gigi[2)], F. Glege[2)], R. Gomez-Reino[2)], C. Hartl[2)], A. Holzner[4)], L. Masetti[2)], F. Meijers[2)], E. Meschi[2)], R. K. Mommsen[5)], C. Nunez-Barranco[2)], V. O'Dell[5)], L. Orsini[2)], C. Paus[6)], A. Petrucci[2)], M. Pieri[4)], G. Polese[2)], A. Racz[2)], O. Raginel[6)], H. Sakulin[2)], M. Sani[4)], C. Schwick[2)], A C Spataru[2)], F. Stoeckli[6)],K Sumorok[6)]

Abstract

This paper outlines a software architecture where zero-copy operations are used comprehensively at every processing point from the Application layer to the Physical layer. The proposed architecture is being used during feasibility studies on advanced networking technologies for the CMS experiment at CERN. The design relies on a homogeneous peer-to-peer message passing system, which is built around memory pool caches allowing efficient and deterministic latency handling of messages of any size through the different software layers. In this scheme portable distributed applications can be programmed to process input to output operations by mere pointer arithmetic and DMA operations only. The approach combined with the open fabric protocol stack (OFED) allows one to attain near wire-speed message transfer at application level. The architecture supports full portability of user applications by encapsulating the protocol details and network into modular peer transport services whereas a transparent replacement of the underlying protocol facilitates deployment of several network technologies like Gigabit Ethernet, Myrinet, Infiniband etc. Therefore, this solution provides a protocol-independent communication framework and prevents having to deal with potentially difficult couplings when the underlying communication infrastructure is changed. We demonstrate the

[1)] DESY, Hamburg, Germany

[2)] CERN, Geneva, Switzerland

[3)] University of California, Los Angeles, Los Angeles, California, USA

[4)] University of California, San Diego, San Diego, California, USA

[5)] FNAL, Chicago, Illinois, USA

[6)] Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

feasibility of this approach by giving efficiency and performance measurements of the software in the context of the CMS distributed event building studies.

Presented at *RT2012: 18th IEEE NPSS Real Time Conference*

A Comprehensive Zero-copy Architecture for High Performance Distributed Data Acquisition over Advanced Network Technologies for the CMS Experiment

Gerry Bauer, Ulf Behrens, James Branson, Sebastian Bukowiec, *Member, IEEE*, Olivier Chaze, Sergio Cittolin, Jose Antonio Coarasa, Christian Deldicque, Marc Dobson, Aymeric Dupont, Samim Erhan, Dominique Gigi, Frank Glege, Robert Gomez-Reino, Christian Hartl, Andre Holzner, Lorenzo Masetti, Frans Meijers, Emilio Meschi, Remigius K. Mommsen, Carlos Nunez-Barranco-Fernandez, Vivian O'Dell, Luciano Orsini, Christoph Paus, Andrea Petrucci, Marco Pieri, Giovanni Polese, Attila Racz, Olivier Raginel, Hannes Sakulin, *Member, IEEE*, Matteo Sani, Christoph Schwick , Andrei Cristian Spataru, Fabian Stoeckli and Konstanty Sumorok

***Abstract*–This paper outlines a software architecture where zero-copy operations are used comprehensively at every processing point from the Application layer to the Physical layer. The proposed architecture is being used during feasibility studies on advanced networking technologies for the CMS experiment at CERN. The design relies on a homogeneous peer-to-peer message passing system, which is built around memory pool caches allowing efficient and deterministic latency handling of messages of any size through the different software layers. In this scheme portable distributed applications can be programmed to process input to output operations by mere pointer arithmetic and DMA operations only. The approach combined with the open fabric protocol stack (OFED) allows one to attain near wire-speed message transfer at application level. The architecture supports full portability of user applications by encapsulating the protocol details and network into modular peer transport services whereas a transparent replacement of the underlying protocol facilitates deployment of several network technologies like Gigabit Ethernet, Myrinet, Infiniband etc. Therefore, this solution provides a protocol-independent communication framework and prevents having to deal with potentially difficult couplings when the underlying communication infrastructure is changed. We demonstrate the feasibility of this approach by giving efficiency and performance measurements of the software in the context of the CMS distributed event building studies.**

Manuscript received June 15, 2012. This work was supported in part by the DOE and NSF (USA) and the Marie Curie Program.

U. Behrens is with DESY, Hamburg, Germany.

S. Bukowiec, O. Chaze, J. A. Coarasa, C. Deldicque, M. Dobson, A. Dupont, D. Gigi, F. Glege, R. Gomez-Reino, C. Hartl, L. Masetti, F. Meijers, E. Meschi, C. Nunez-Barranco-Fernandez, L. Orsini, A. Petrucci (corresponding author. phone: +41 22 76 70808, fax: +41 22 76 69051, e-mail: Andrea.Petrucci@cern.ch), G. Polese, A. Racz, H. Sakulin, C. Schwick, and A. C. Spataru are with CERN, Geneva, Switzerland.

S. Erhan is with University of California, Los Angeles, California, USA.

J. Branson, S. Cittolin, A. Holzner, M. Pieri and M. Sani are with University of California San Diego, La Jolla, California, USA.

R. K. Mommsen and V. O'Dell are with FNAL, Batavia, Illinois, USA.

G. Bauer, C. Paus, O. Raginel, F. Stoeckli and K. Sumorok are with Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

I. Introduction

THE Compact Muon Solenoid (CMS) [1] is a general-purpose particle detector at the Large Hadron Collider (LHC) [2] at CERN in Geneva, Switzerland.

TABLE I. NOMINAL PARAMETERS OF THE CMS DAQ.

Parameter	Value
Beam crossing rate	40 MHz
Level-1 Trigger rate	100 kHz
Number of front-ends	700
Event Size	1 MByte
Event Builder throughput	100 GByte/s
Maximum rate after HLT	O(100) Hz

Table I shows the main parameters of the Trigger and Data Acquisition (TriDAS) system in the case of proton-proton collisions at the design LHC luminosity of 10^{34} $cm^{-2}s^{-1}$. A rejection power of $O(10^5)$ is require in order to reduce the event rate from the 40 MHz LHC beam crossing to an acceptable rate of O(100) Hz for physics analysis.

Online event-selection is done using two trigger levels: a hardware-based first-level trigger and a software-based high-level trigger (HLT). Fig. 1 shows the CMS Data Acquisition (DAQ) [3] architecture. The system is designed to read out event fragments of an average size of up to 2 kB from around 700 detector Front-Ends Drivers (FEDs) at a rate of 100 kHz. For FEDs with smaller fragment size, the Frontend Readout Link (FRL) reads out two FEDs and merges the fragments in order to balance fragment sizes. Events are built in two stages: FED Builder and RU Builder. The first stage is the FED Builder and it receives ~500 event fragments coming from the FRLs. Readout Units (RUs) of parallel RU Builder slices assembles groups of 8 event fragments into super-fragments. In each slice there are 72 RUs connected to the FED Builders, which send the received data to the Builder Units. The Builder Units (~125 for each slice) build and analyze the full events and forward the selected events to mass storage. The DAQ is

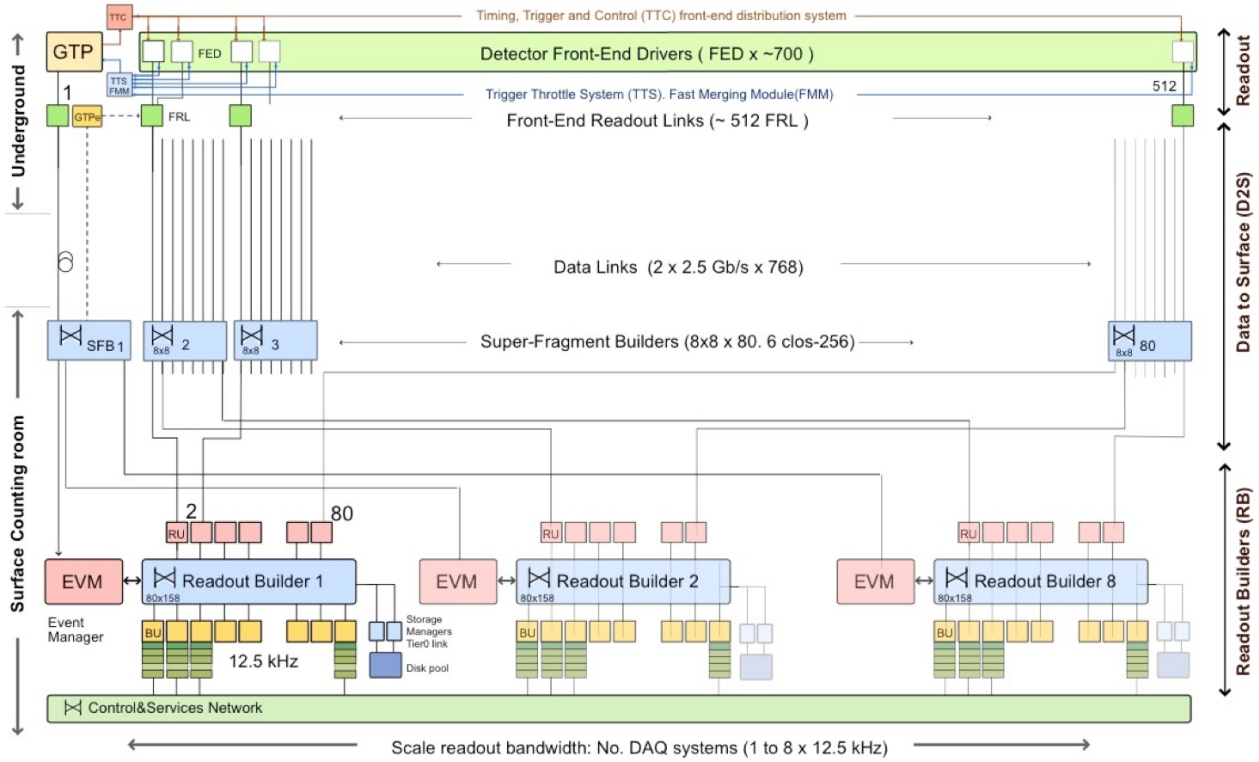


Fig. 1. The CMS data acquisition architecture. Events are built in two stages: super-fragments are built in the Readout Unit (RU), full events are built in the Builder Unit (BU). The BUs also run the high-level trigger software.

composed of few thousand hosts [4][5] and of O(20000) interdependent applications.

The online applications are based on the XDAQ [6] framework that is a software platform designed specifically for the development of distributed data acquisition systems. The framework is a software middleware that eases the tasks of designing, programming and managing data acquisition applications by providing a simple, consistent and integrated distributed programming environment. XDAQ builds upon industrial standards, open protocols and libraries.

Through the XDAQ software, CMS has successfully been recording proton-proton collisions at a center-of-mass energy of 7 TeV during 2010 and 2011 and at 8 TeV since the start of 2012. A long shutdown is planned from 2013 to September 2014 in order to upgrade the LHC machine for reaching the luminosity of 2x 10^{34} $cm^{-2}s^{-1}$ at 25 ns or 50 ns bunch spacing. During the long shutdown some CMS sub-detector front-end electronics and readout systems will be upgraded, and a new L1 trigger system will be deployed and operated in parallel to the existing system. Aging of existing hardware (PCs and networks at least 5 years old) and to accommodate sub-detectors with upgraded off-detector electronics are the main motivations for the upgrade of CMS DAQ system. The upgrade plan for the DAQ system is to replace FED builder and RU builder networks with more recent network technologies. The DAQ team has started feasibility studies on advanced networking technologies to identify the network technology to use for the upgrade of the event builder network. In the present paper we describe the architecture of XDAQ, and the integration of RDMA-capable transports within the framework by means of uDAPL, the User Direct Access Programming Library [7]. We report on preliminary results of the 10 Gigabit Ethernet (10 GbE) and 4x QDR Infiniband performance tests.

II. XDAQ Framework

The XDAQ distributed programming environment follows a layered middleware approach [8], designed according to the object-oriented model and implemented using the C++ programming language [9]. The distributed processing infrastructure is made scalable by the ability to partition applications into smaller functional units that can be distributed over multiple processing units. In this scheme each computing node runs a copy of an executive that can be extended at run-time with binary plugin components. The program exposes two types of interfaces: “core” and “application”. The core interfaces lie between the middleware and core plugin components, providing access to basic system functionalities and communication hardware. Core plugins manage basic system functions on behalf of the user applications, including network access, memory management and device access. The application interfaces provide access to the various functions offered by the core plugins and are placed between the middleware and the user application components as shown in Fig. 2.

Middleware services include information dispatching to applications, data transmission, exception handling facilities, access to configuration parameters, and location lookup (address resolution) of applications and services. Other system services include locking, synchronization, task execution and memory management. Applications communicate with each other through the services provided by the executive according

to a peer-to-peer message-passing model. This allows each application to act both as a client and a server.

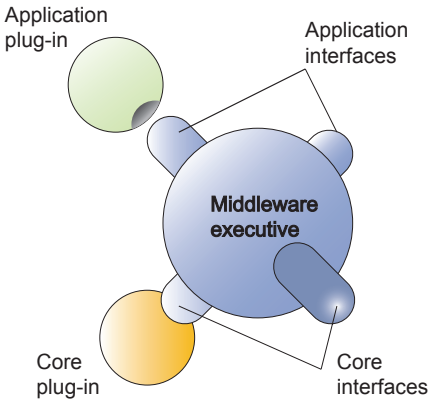


Fig. 2. Middleware interfaces.

The general programming model follows the event driven processing scheme [10] where an event is an occurrence within the system. It can be an incoming message, an interrupt, the completion of an instruction, like a direct memory access transfer, or an exception. Messages are sent asynchronously and trigger the activation of user-supplied callback procedures when they arrive at the receiver side. Every event therefore corresponds to a message that follows a standardized format.

The event-based processing model has been chosen because it is known to scale [13]. There is no need for a central place in which the incoming data have to be interpreted completely. It is the responsibility of each software component that listens to a given type of event (e.g. data received, timeout) to decide what it should do with the information received. Extensibility is thus achieved through the decoupling of the reception of a message and its processing. The procedure for a given message can be provided dynamically during run-time by downloading a software module that contains all code to react to an incoming message of a given type. Furthermore it is possible to add new functionality by defining new messages. The system provides a default procedure if for a given event no executable function has been supplied. This model results in a homogeneous structure of software components with an intrinsic fault-tolerant behavior.

The software distribution comes with two ready to use communication protocols. One is based on the I_2O specification [11] used for efficient and high performance data transmission, and the other on SOAP and XML [12] used for configuration purposes.

A. Memory Management

The executive program provides applications with efficient memory management facilities. These are based on a scheme called "buffer-loaning" which avoids fragmentation of memory over long run periods and presents a safe operation model that prevents extensive growth of memory consumption. With the buffer-loaning scheme, applications or core-plugins ask the executive for fixed-sized chunks of memory from one of various buffer pools. The principle is displayed graphically in Fig. 3.

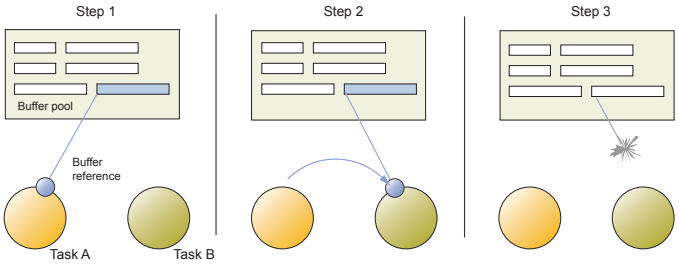


Fig. 3. Illustration of the buffer-loaning mechanism. A task loans a reference to an unused buffer that matches the closest requested data size from a buffer pool (step 1). The buffer can be passed to another task by forwarding the buffer reference (step 2) without copying the data. The buffer is released to the pool by destroying the buffer reference (step 3). It can now be re-allocated to another task.

The executive manages various types of pools, including ordinary user-space memory, a reserved amount of physical memory (e.g. the Linux "bigphys" kernel extension) and memory on network cards. The pools can be configured such that exceptions are raised if too little memory is available. All memory buffers are allocated from the available pool and are accessible through a reference that can be further lent to other software components.

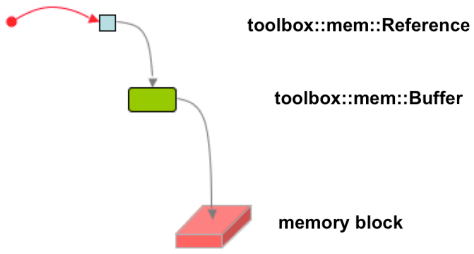


Fig. 4. The toolbox::mem::Reference structure is the primary vehicle through which XDAQ applications access or pass the data that resides in a memory pools established by the communicating applications.

The executive routines deal with data in terms of memory buffer references *toolbox::mem::References* and abstract memory buffers *toolbox::mem::Buffer* for various kinds of memories (see Fig. 4).

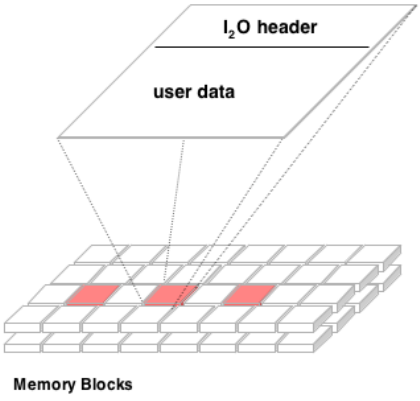


Fig. 5. I2O frame memory blocks.

For example, a user application prepares a message and passes it on to a transport component that handles the network transmission. Eventually, the buffer must be returned to its pool. Built-in reference counting ensures that a buffer is not returned into its originating pool before the very last user has released it.

XDAQ uses the *toolbox::mem::Reference* and the *toolbox::mem::Buffer* structures to track information necessary to manage the data in the I_2O frames (see Fig. 5). A much more detailed description of I_2O protocol and data format can be found in section II.C.

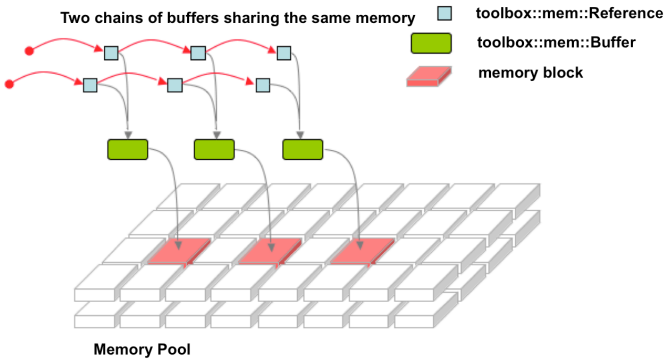


Fig. 6. Chaining is one of the *toolbox::mem::Reference* feature. This lets XDAQ applications pass an arbitrarily large amount of data by passing the *toolbox::mem::Reference* at the head of a *toolbox::mem::Reference* chain.

Various buffers can be chained together to allow arbitrarily sized data (see Fig. 6). The mechanism not only enables efficient zero-copy implementations, but also provides the foundation for transparent operation across network boundaries. Various high-speed interconnects and custom built electronics rely on non-standard memory models, that would otherwise require the instrumentation of user programs with special instructions. Buffer pools can be added to the executive to offer specific allocation through an interface that is common to all memory types.

B. *Data Transmission*

A message between two endpoints that are located on two different hosts might need to travel through a media. To determine how a message should be sent between two endpoints, a mechanism is required to allow a peer to discover the route information. The XDAQ framework provides peers with a mechanism for determining a route to an endpoint, allowing the peer to send data to the remote endpoint. Peer transports (see Fig. 7) are the entity responsible for conducting the actual exchange of information over a network. Peer transport encapsulate a set of network interfaces, allowing a peer to send and receiver data independently of the type of network being employed.

Communication between ordinary applications is accomplished by means of an executive function. This function, when invoked, re-directs the outgoing message to the proper peer-transport that in turn delivers the data over the associated medium. In this way the framework is independent of any transport protocol or network and can be extended at any time to accommodate newly appearing communication technologies.

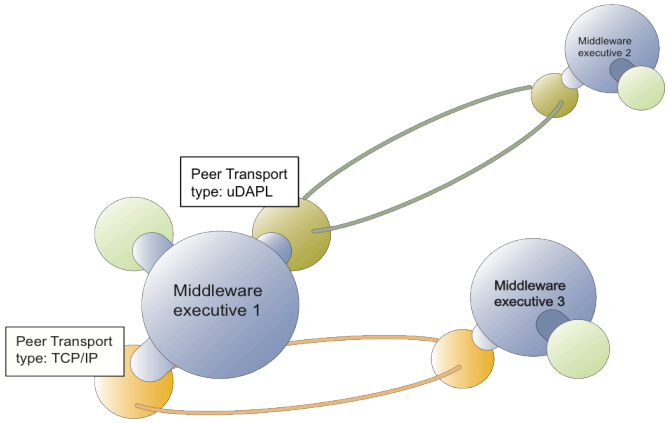


Fig. 7. Communication over multiple networks through peer-transports.

XDAQ offers two simple ways to send and receive messages: the Application Context Services and the Peer Transport Agent services. The Application Context Services are simply convenient wrappers for sending and receiving messages using a peer transport. A Peer Transport is an interface to a set of network transports that allows data to be sent across the network. Details on how data is to be formatted for transport across the network is the responsibility of a particular peer transport implementation.

C. *I_2O Protocol and Data Format*

The framework supports the exchange of messages using the I_2O binary data format. I_2O (Intelligent Input Output) is originally a specification for an I/O architecture developed by a consortium of computer companies called the I_2O special Interest Group (SIG). I_2O is designed to eliminate I/O bottlenecks by utilizing special I/O processors (IOPs). In particular, I_2O was also designed to facilitate intelligent I/O subsystems, with support for message-passing between multiple independent processors. This concept includes therefore a communication scheme for the data exchange among devices with processing capabilities namely Peer-to-Peer message passing. The I_2O Peer to Peer message passing as supported in XDAQ relies on three key properties:

- Independence of the used transport protocol;
- Asynchronous communication in connection with a callback engine;
- A common data format for all messages.

I_2O messages are datagrams with a maximum size of 256 kB. For sizes larger than this maximum, the data have to be split and sent in a sequence of multiple frames. The common data format, as outlined in Fig. 8, encapsulates the destination identifier of the application that shall receive the message. The callback function that is invoked upon receive can be one of the standard I_2O functions (Number is field Function ID), or a user supplied callback (Function ID = 0xFF and the extended header contains a numeric identifier in XFunction). The message also contains the sender identifier, the originator. Any user data can be inserted into the message after the extended header.

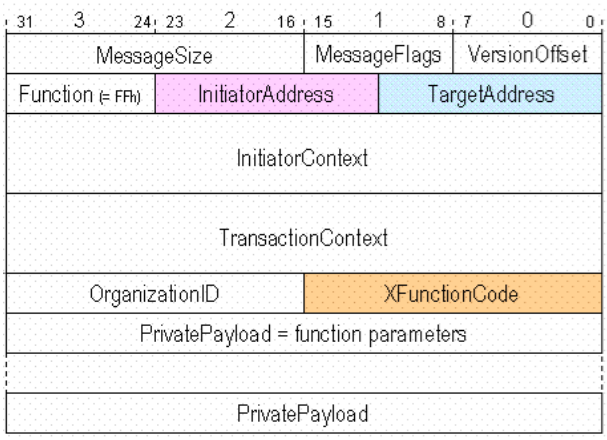


Fig. 8. I_2O header format.

I_2O messages are declared as C structures and are therefore statically defined at compile time. Any modification of the message structure requires the adaptation of the sender and the receiver, respectively. I_2O defines, that the ordering of bytes on the network is Little Endian, aligned to 32 or 64 bit boundaries.

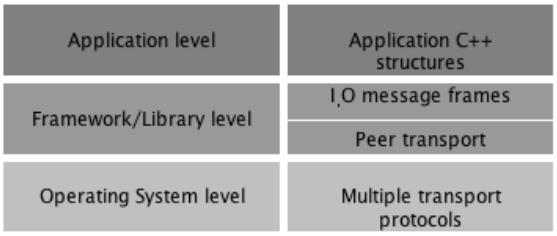


Fig. 9. The interaction of the layers in XDAQ framework.

XDAQ performs the necessary conversion for the message header if messages are exchanged between machines of different native byte ordering. Asynchronous communication means that the sender will not block on waiting for successful reception of the message at the receiver side.

I_2O is used as an application level protocol (see Fig. 9). I_2O frames as shown above can be exchanged among XDAQ applications over several different transport protocols. The choice of communication protocol is made at configuration time through the selection of loadable peer-transport components, which implement the network dependent communication mechanisms. Thus, the application code remains invariant with respect to the various network technologies. This interface implements a buffer-loaning scheme where memory segments are exchanged among all components within the framework. With all the above mechanisms, it is therefore possible to confine data transmission for input and output to pure DMA operations and pointer arithmetics.

III. DAPL Integration

In the last years, as the network speed has increased toward 100 gigabits per second, the CPU must spend more time working to service the network. To process network requests for the de facto networking standard, TCP/IP, the processor must dedicate a large number of cycles and resources to data transfers. To avoid this problem, technology such as Infiniband [19], iWARP [20], and RDMA over Converged Ethernet (RoCE) [21] have been developed that not only allow for a very fast interconnect, but also provide a mechanism known as Remote Direct Memory Access (RDMA) [22] to bypass the operating system and CPU in order to directly move data into application memory.

The OpenFabrics Alliance (OFA) [23] develops, tests, licenses, supports and distributes OpenFabrics Enterprise Distribution (OFED™) [24] open source software to deliver high-efficiency computing, wire-speed messaging, ultra-low microsecond latencies and fast I/O. The goal of the OpenFabrics Alliance is to deliver a unified, cross-platform, transport-independent software stack for RDMA and kernel bypass. Transport independence means that users can utilize the same OpenFabrics RDMA and kernel bypass API and run their applications agnostically over Infiniband, iWARP or RoCE. The OFED is the software on the host that coordinates user-space and kernel-space access to the Infiniband or Ethernet hardware.

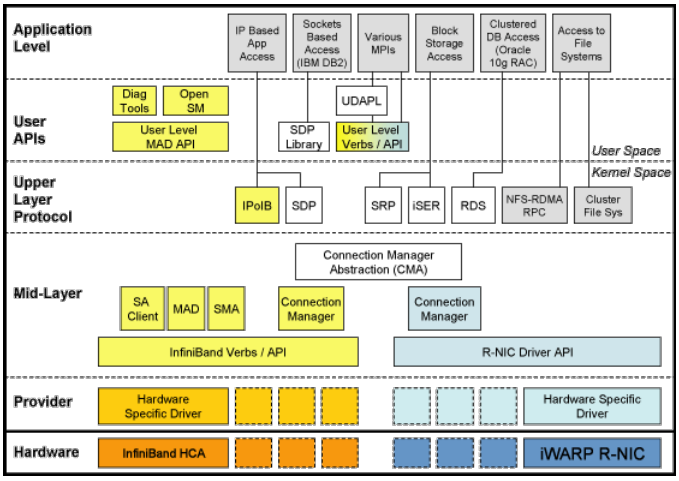


Fig. 10. The OFED Stack (source: OpenFabrics Alliance)

As shown Fig. 10, the OFED stack consists of many different components. These components can be categorized as kernel modules (drivers) and user/system libraries and utilities, commands and daemons for Infiniband or Ethernet administration, configuration, and diagnostics. The Direct Access Programming Library (DAPL) [25] developed by DAT collaborative is a distributed messaging technology that is both hardware-independent and compatible with current network interconnects. The architecture provides an API that can be utilized to provide high-speed and low-latency communications among peers in clustered applications. The DAT Collaborative's goal is to define the interface between uDAPL Provider (DAT-compliant Interface Adapter driver) and DAT Consumer (Application). As shown in Fig. 11, uDAPL defines the API for the kernel level when uDAPL Provider is within OS and below, while DAPL defines the API for the user level when DAT Consumer is completely within application space. Each Interface Adapter is controlled by exactly one uDAPL Provider. Each uDAPL Provider can control multiple Interface Adapters. There can be multiple

DAT Providers controlling disjoint sets of Interface Adapters on a host.

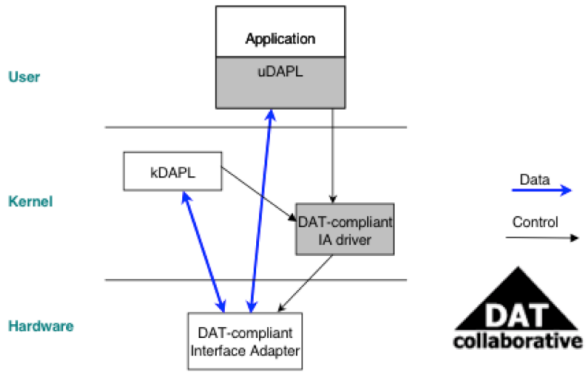


Fig. 11. DAT User Architecture Model. The diagram shows the DAT kernel API architecture model, including uDAPL Consumer, uDAPL Provider, OS, and Interface Adapter.

A new peer transport (ptuDAPL) has been implemented for DAT library using I_2O messaging protocol and based on DAT Specification 2.0 [26].

Fig. 12. ptuDAPL integration with XDAQ framework.

It uses smart memory pool based on uDAPL memory region allocator, and the random access to memory with no intermediate management is performed using cookies. All I/O operations are centered on dedicated uDAPL memory pool that allows a full zero-copy between XDAQ applications and DAPL driver. The API is optimized to minimize the latency profiting for inherent non-blocking and queuing of uDAPL. As shown in Fig. 12, the ptuDAPL can be integrated in the XDAQ framework without change of application code.

IV. Cluster Setup

To perform benchmark evaluation of the new peer transport ptuDAPL, we used a small cluster. The setup consisted in 8 nodes of DELL PowerEdge R710 with dual sockets Intel Xeon E5530 4-core at 2.27 GHz and 3GB of memory. The operating system running on the nodes was Scientific Linux CERN 5 (SLC5) with the 2.6.18-164.6.1.el5 kernel. Each node was equipped with an Infiniband Host Channel Adapter (HCA) supporting 4x Quad Data Rate (QDR) connections with data rate of 32Gbps (Qlogic HCA, qle7340 4x QDR PCIe), and iWARP adapter at 10 GbE (Chelsio T420-CR 10GBASE-SFP RNIC). Each node was connected with an Infiniband switch (Qlogic 12300-BS01- 4x QDR) and 10 GbE switch (Voltaire Vantage 6048).

V. Benchmarks

To evaluate the different network technologies we have done three different benchmarks:

- Latency of sending a packet between two nodes using ptuDAPL to measure the overhead of the XDAQ framework;
- Measurement of the maximum throughput per node between one node to more nodes with a multi-streams of I/O data;
- Measurement of the maximum throughput per node between N nodes to N nodes with event builder software.

A. Latency Measurements

We developed a XDAQ application called *roundtrip* to measurement the latency of sending a package between two nodes. A time packet is travelling from a specific source to a specific destination and back again; one-way latency is measured by timing a round-trip message and dividing the obtained result by two (see Fig. 13).

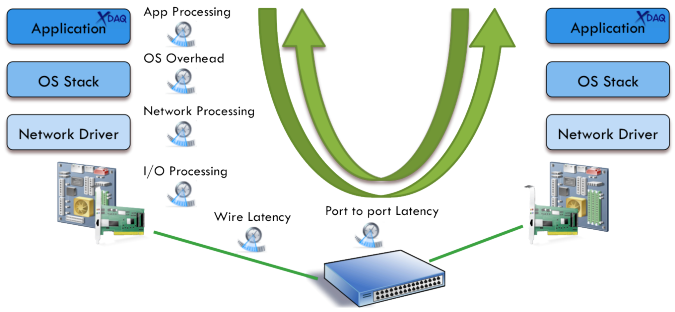


Fig. 13. Diagram of latency factors.

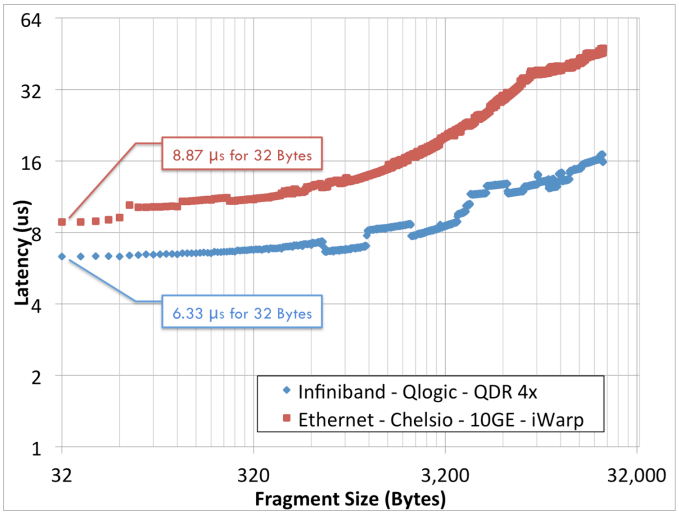


Fig. 14. Latency in us versus fragment size in Bytes for Infiniband - 4x QDR - Qlogic (blue points) and Ethernet - iWARP - Chelsio (red points). The series with latency and fragment size correspond to log normal distributions.

Fig. 14 shows the latency for sending a package with different fragment size using ptuDALP over Infiniband (4x

QDR) and Ethernet (iWARP). For a packet of 32 Bytes the overhead of XDAQ framework is less then 1 us for both network technologies.

B. *Maximum Throughput per Node with Stream of I/O Data*

In order to calculate the maximum throughput per node we implemented a XDAQ application (Multi-Stream I/O) to send multiple streams I/O data from one source to many destinations. As shown in Fig. 15, throughput per node is measured sending continuously N messages to N receivers and time sampling is done at the receiver's side.

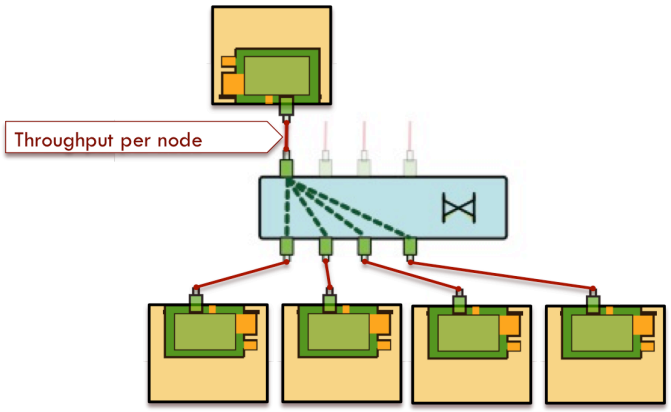
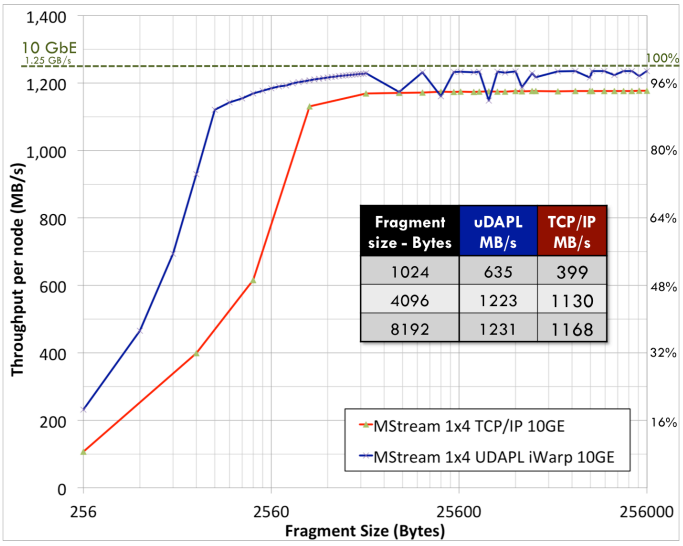


Fig. 15. Diagram of the multi-stream I/O application.



Fragment size - Bytes	uDAPL MB/s	TCP/IP MB/s
1024	635	399
4096	1223	1130
8192	1231	1168

Fig. 16.Throughput per node in MB/s using multi-stream I/O application versus fragment size in Bytes for uDAPL/iWARP (blue line) and TCP/IP (red line). The series fragment size corresponds to log normal distributions.

A configuration with one sender and four receivers has been tested using uDAPL/iWARP versus TCP/IP. The throughput per node as a function of fragment size is shown in Fig. 16. In the case of uDAPL/iWARP, it reaches a plateau of about 1200 MB/s for packet sizes above 3 kB with an efficiency close to 100%. The performance for TCP/IP, is also shown in Fig. 16, and it can be seen that the throughput per node is less than uDAPL/iWARP with a considerable difference for small fragments, as expected for the cost of TCP/IP stack.

C. *Maximum Throughput per Node with Event Builder*

To perform the maximum throughput per node with the event builder application we used the CMS RU-Builder [28] software in emulation mode. RUs generate the event fragment data and BUs discard the event data once an event is fully assembled. The L1 trigger is not emulated and all measurements correspond to the saturation limit. Fig. 17 shows the event builder protocol. With free capacity available, a BU requests the EVM to allocate it an event (step 1). The EVM confirms the allocation by sending the BU the event ID and trigger data of an event (step 2). This trigger data is the first super-fragment of the event. The BU now requests the RUs to send it the rest of the event's super-fragments (step 3). The BU builds the super-fragments it receives from the RUs (step 4) into a whole event within its resource table (step 5). FUs can ask a BU to allocate them events (step 6). A BU services a FU request by sending the FU a whole event (step 7). When a FU has finished with an event, it tells the BU to discard it (step 8).

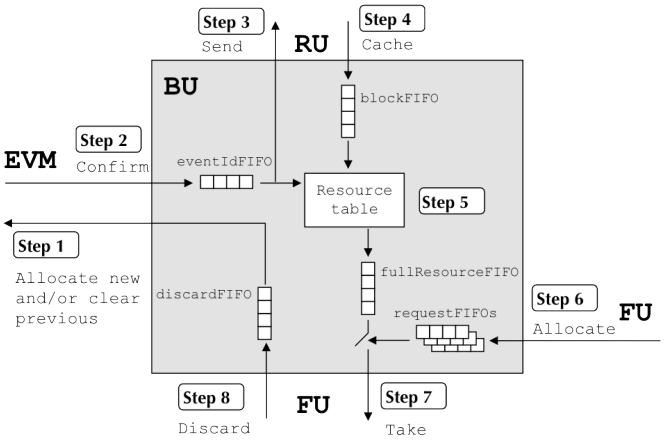
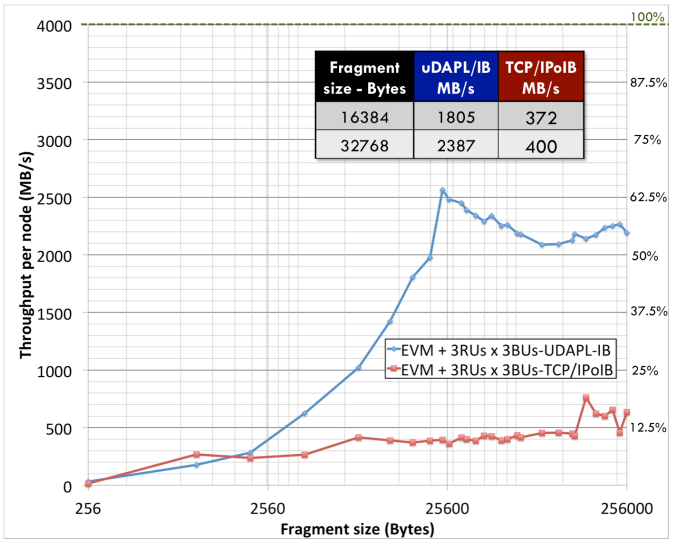


Fig. 17. Event builder protocol.



Fragment size - Bytes	uDAPL/IB MB/s	TCP/IPoIB MB/s
16384	1805	372
32768	2387	400

Fig. 18. Throughput per node in MB/s using event builder application versus fragment size in Bytes for uDAPL/IB (blue line) and TCP/IPoIB (red line). The series fragment size corresponds to log normal distributions.

An event builder configuration with an EVM, 3 RUs and 3 BUs has been tested using uDAPL/IB versus TCP/IPoIB (4x QDR). The throughput per node as a function of fragment size is shown in Fig. 18. In the case of uDAPL/IB, it reaches a plateau of about 2 GB/s for sizes above 20 kB with an efficiency ~ 55%. The efficiency of Input-Queued Switches with random traffic (no traffic shaping) is $\varepsilon=2-\sqrt{2} \approx 0.59$ for $N \rightarrow \infty$ [27]. The performance for TCP/IPoIB is very low compered to uDAPL/IB.

VI. Summary

In this paper we have shown the XDAQ architecture and the integration of RDMA-capable transports within the framework by means of uDAPL. The new ptuDAPL provides a protocol-independent communication framework and avoids any potential problem when the underlying communication infrastructure changes. The preliminary tests have given interesting results: uDAPL/iWARP over 10 GbE shows a better throughput per node for small fragment sizes as compared to the traditional TCP/IP stack on the host, in Infiniband we saw that the TCP/IPoIB gives only ~ 12 % of efficiency.

To continue our feasibility studies for the CMS event builder we need a bigger cluster to check the scalability. We are setting up a new system with 32 nodes of DELL PowerEdge C6220 with dual sockets Xeon E5-2670 8-core at 2.6 GHz and 32GB of memory. Each node is equipped with a Mellanox - ConnectX-3 VPI adapter (MCX353A-FCBT) supporting 4x Fourteen Data Rate (FDR) connections with data rate of 54.4 Gbps and 40 GbE. Using the new setup we can perform scalability tests, try to improve the Infiniband efficiency using the Quality of Service and test RoCE technology.

Acknowledgment

This work was supported in part by the DOE and NSF (USA) and the Marie Curie Program.

References

[1] The CMS Collaboration, The Compact Muon Solenoid Technical Proposal, CERN/LHCC94-38 (1994)

[2] The LHC Study Group, The Large Hadron Collider Conceptual Design Report, CERN/AC95-05 (1995).

[3] The CMS Collaboration, The Trigger and Data Aquisition project, CERN/LHCC 2002-26, 15 December 2002.

[4] Antchev G et al 2001, The Data Acquisition System for the CMS Experiment at LHC *in Proc. 7th Intl. Conf. Adv. Tech. and Particle Phys. Villa Olmo, Como, Italy (Oct. 15-19, 2001)* World Scientific Publishers (ISBN 981-238-180-5)

[5] Bauer G et al 2008 *CMS DAQ Event Builder Based on Gigabit Ethernet* IEEE Trans. Nucl. Sci. **55(1)** 198-202

[6] Bauer G et al., "The CMS data acquisition system software," *J. Phys.: Conf. Ser.* 219 022011,2010

[7] User-Level Direct Access Transport APIs (uDAPL), http://www.datcollaborative.org/udapl.html

[8] P. Bernstein, "Middleware: A Model for Distributed Systems Services", Communications of the ACM, 39 (1996) 86.

[9] J. Gutleber and L. Orsini, "Software Architecture for Processing Clusters Based on I2O", Cluster Computing 5(1):55-65, Kluwer Academic Publishers, 2002.

[10] B.N. Bershad et al.,"Extensibility, Safety and Performance in the SPIN Operating System", in Proceedings of the Fifteenth ACM Symposium on Operating System Principles, pp. 267-284, 1995.

[11] I2O Special Interest Group, Intelligent I/O (I2O) Architecture Specification v2.0, 1999, at http://www.intelligent-io.com

[12] J. Boyer, "Canonical XML Version 1.0, W3C Recommendation 15 March 2001", http://www.w3.org/TR/xml-c14n

[13] P. Pardyak and B.N. Bershad, "Dynamic binding for an extensible system", in Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation, 1996, pp. 201-212.

[14] D. Box et al.," Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000", http://www.w3.org/TR/SOAP

[15] R. Fielding et al.,"Hypertext Transfer Protocol - HTTP /1.1", IETF RFC 2616, June 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html

[16] G. Glass, "Web Services: building blocks for distributed systems", Prentice Hall, 2002.

[17] "Java API for XML Messaging (JAXM) Specification v1.0", Sun Microsystems, 901 San Antonio Road, Palo Alto, California 94303, USA, October 2001.

[18] A. Le Hors et al.,"Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Working Draft 13, September 2001", http://www.w3.org/TR/2001/WD-DOM-Level-3-Core-20010913

[19] InfiniBand Architecture Specification, InfiniBand Trade Association, October 2004. URL http://www.infinibandta.org/specs/

[20] H. Shah et al. (October 2007). "Direct Data Placement over Reliable Transports". *RFC 5041*. http://tools.ietf.org/html/rfc5041

[21] InfiniBand Trade Association, *InfiniBand™ Architecture Specification Release 1.2.1 Annex A16: RoCE*, InfiniBand Trade Association, April 2010. http://members.infinibandta.org/kwspub/spec/Annex_RoCE_final.pdf

[22] Jeff Hilland, Paul Culley, Jim Pinkerton, and Renato Recio. RDMA Protocol Verbs Specification, April 2003. http://www.rdmaconsortium.org/home/draft-hilland-iWARP-verbs-v1.0-RDMAC.pdf

[23] OpenFabrics Alliance, www.openfabrics.org/

[24] OpenFabrics Enterprise Distribution (OFED™), https://www.openfabrics.org/ofed-for-linux-ofed-for-windows/ofed-overview.html

[25] Direct Access Programming Library (DAPL), http://www.datcollaborative.org/

[26] uDAPL API Spec Version 2.0, http://www.datcollaborative.org/uDAPL_v20.zip

[27] *M. J. Karol*, *M. G. Hluchyj, and S. P. Morgan*, "*Input Versus Output* Queueing on a. Space-Division Packet Switch," IEEE Transactions on Communications, vol. 2, pp. 277-287, 1989.

[28] G. Bauer et al., The CMS event builder and storage system, 17th International Conference on Computing in High Energy and Nuclear Physics (CHEP 09), DOI: 10.1088/1742-6596/219/2/022038