Boosting Event Building Performance Using Infiniband FDR for CMS Upgrade

Tomasz Bawej[b], Ulf Behrens[a], James Branson[d], Olivier Chaze[b], Sergio Cittolin[d], Georgiana-Lavinia Darlea[f], Christian Deldicque[b], Marc Dobson[b], Aymeric Dupont[b], Samim Erhan[c], Andrew Forrest[1b], Dominique Gigi[b], Frank Glege[b], Guillelmo Gomez-Ceballos[f], Robert Gomez-Reino[b], Jeroen Hegeman[b], Andre Holzner[d], Lorenzo Masetti[b], Frans Meijers[b], Emilio Meschi[b], Remigius K. Mommsen[e], Srecko Morovic[b], Carlos Nunez-Barranco-Fernandez[b], Vivian O'Dell[e], Luciano Orsini[b], Christoph Paus[f], Andrea Petrucci[b], Marco Pieri[d], Attila Racz[b], Hannes Sakulin[b], Christoph Schwick[b], Benjamin Stieger[b], Konstanty Sumorok[f], Jan Veverka[f], Petr Zejdl[b]

a. DESY, Hamburg, Germany
b. CERN, Geneva, Switzerland
c. University of California, Los Angeles, Los Angeles, California, USA
d. University of California, San Diego, San Diego, California, USA
e. FNAL, Chicago, Illinois, USA
f. Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

E-mail: `aforrest@cern.ch`

As part of the CMS upgrade during CERN's shutdown period (LS1), the CMS data acquisition system is incorporating Infiniband FDR technology to boost event-building performance for operation from 2015 onwards. Infiniband promises to provide substantial increase in data transmission speeds compared to the older 1GE network used during the 2009-2013 LHC run. Several options exist to end user developers when choosing a foundation for software upgrades, including the uDAPL (DAT Collaborative) and Infiniband verbs libraries (OFED). Due to advances in technology, the CMS data acquisition system will be able to achieve the required throughput of 100 kHz with increased event sizes while downsizing the number of nodes by using a combination of 10GE, 40GE and 56 Gb Infiniband FDR. This paper presents the analysis and results of a comparison between GE and Infiniband solutions as well as a look at how they integrate into an event building architecture, while preserving the scalability, efficiency and deterministic latency expected in a high end data acquisition network.

Technology and Instrumentation in Particle Physics 2014
2-6 June, 2014
Amsterdam, the Netherlands

1 Speaker

1.Introduction

The proficiency of distributed event building is often characterized by several factors, such as its choice of network switches and links, memory efficiency and processor speed etc. In the context of event building for the Compact Muon Solenoid (CMS) upgrade [1][2][3], increased processing power per node and faster link speeds opens the opportunity to design solutions where the required event-building performance can be achieved using a smaller number of physical computing and networking resources. This has numerous ramifications for both the hardware and software that support the network transmission. In order to fully exploit the introduction of new networking hardware, a well-suited software architecture must be chosen to dramatically reduce overhead and latency introduced by the available protocol software layers. A common example of such an overhead in traditional or optimized TCP/IP implementations is the implicit copying of data from the NIC receive buffer several times before being delivered to the destination application. In addition, within the socket programming abstraction commonly used for network communication, the conversion of buffers into streams and vice-versa adds an unnecessary complication to an application like an event builder that needs to manipulate and exchange buffers containing event data. A network programming abstraction that allows the transfer of blocks of memory directly (zero-copy) from source to destination is a better fit when designing event-building software that should be capable of reaching full network throughput. Infiniband [4] is such an interconnect which supports a message based send and receive semantic with a fully thread safe, zero-copy, asynchronous event-based architecture for network communication.

For the LS1 upgrade, CMS has investigated and subsequently selected Infiniband as the network fabric for the event builder network. The DAQ group has been able to evaluate both the ibverbs (OFED) [5] and the uDAPL (DAT Collaborative) [6] technologies. The choice over which direction to use when integrating new technologies into a system is based upon the system's requirements with respect to common software aspects, such as performance, usability, scalability and reliability. This paper will present the findings of a feasibility study with regard to how each approach meets these qualities. The Infiniband technologies were integrated into the CMS Online Software XDAQ framework [7], permitting us to demonstrate the effective enhancement in performance for the existing CMS event builder while reducing the number of physical resources by approximately one order of magnitude.

2. CMS Data Acquisition System for LHC Run 2

The CMS is a general-purpose particle detector that is used to study collisions of protons and heavy ions produced by the LHC [8] at CERN in Geneva, Switzerland. To reduce the event rate from 40 MHz (LHC beam crossing frequency) down to a rate of O(1000) Hz which is an acceptable rate for storage and analysis, CMS uses a two level trigger system. The first level trigger is based in hardware and reduces the rate of events to 100 kHz. The second, high-level trigger (HLT) reduces the rate down to O(1000) Hz using software running on commodity of-the-shelf hardware. This DAQ architecture proved to be successful during LHC run 1, with an availability of more than 99.6 %.

In order to accommodate the increased energy levels of 13~14 TeV and a luminosity of $2x10^{34}$ $cm^{-2}s^{-1}$ that will be used during the LHC run 2, many aspects of the CMS experiment are being upgraded during the long-shutdown 1 (LS1). At the same time, the equipment used for the DAQ during run 1 has reached the end of their 5-year life span and needs to be replaced. In addition, some sub-detectors will have upgraded their off-detector electronics and readout systems based upon the µTCA [9] standard which needs to be supported by the DAQ.

Many important DAQ parameters will not have changed between run 1 and run 2, including the LHC beam crossing rate of 40 MHz and level one trigger rate of 100 kHz. The event size will increase from 1 MB to 2 MB due to additional readout channels and an increase of luminosity. As a result, the readout and event builder must be able to handle a throughput of 200 GB/s, up from 100 GB/s in run 1.

CMS uses a software framework called XDAQ to ease the development of online DAQ applications such as the event builder.

3.Software Architecture

The XDAQ framework follows a layered middleware approach, designed according to the object-oriented model and is implemented in C++. It is designed specifically for development of distributed applications. 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. Applications run within copies of an executive that can be run within a single node or distributed across a cluster. Inter-application communication is achieved through services provided by the executive according to a peer-to-peer message-passing model. The executives are extended at run-time with the required binary plugin components to support any needed additional functionalities, including networking. The general programming model follows the event-driven processing scheme where an event is an occurrence within the system. Communication is through messages that are sent asynchronously and received through user-supplied callback procedures. The design of the framework is such that it allows applications to scale in proportion to available resources. The configuration at run time of a XDAQ application is achieved through the use of standardized XML based configuration files.

3.1 Multithreading

Applications should be designed to transparently benefit from parallelism available on a hardware/software platform such as multi-core or multi-processor systems [10]. XDAQ supports multi-core systems by providing a mechanism to associate threads with specific processing cores, allowing a high granularity of control over where and how processes execute concurrently. This functionality is controlled through the XML configurations.

3.2 Memory Management

The framework provides applications with efficient memory-management facilities, based upon the concepts of memory pools and buffer loaning. The use of memory pools allows fast and deterministic allocation time and avoids fragmentation of memory over long run periods by allocating fixed-sized blocks of memory from one of various buffer pools. Multiple buffers can be chained together to allow arbitrarily sized data. The framework manages various types of memory pools transparently, and hides any additional memory allocation tasks from the user (e.g. the registration of memory with the Infiniband NIC).

XDAQ applications deal with data as references to buffers, which are allocated by a memory pool. These references can be exchanged via a buffer loaning scheme, which enables zero-copy transfer of data through different software layers. When a buffer is no longer needed, the reference to it can be released, at which point ownership is returned to the pool and it becomes available for re-allocation. Additionally, the framework supports Non Uniform Memory Access (NUMA) [11] environments and allows the allocation of memory within a memory pool to be bound to a specific NUMA node for efficient access to memory resources.

3.3 Data Transmission and Format

The XDAQ framework provides an interface to plugins known as Peer Transports that perform the transmission of data between applications in a distributed environment. An endpoint abstraction is used to determine routes for communication within the network for a given protocol. When a message is sent from one application to another, the framework determines the configured route and transmits the data over the corresponding endpoint by matching the endpoint protocol to a peer transport plugin that is used to perform the actual communication. As endpoints are matched to a wire protocol (e.g. TCP), the framework allows multiple protocols to be used at the same time. The combination of endpoints and peer transports allows for applications written within the XDAQ framework to remain network layer and protocol independent, as new network technology plugins can be developed and then deployed by changing just the runtime configuration. The peer transport interface relies on the buffer reference abstraction for accessing data allowing the plugin implementation to maintain a zero-copy architecture throughout. For data flow, XDAQ uses a binary data format based on the I_2O specification [12] at an application level.

4. Event Building in the CMS DAQ

Event building is the process by which data that is distributed across multiple sources (fragments) is collected and merged into a single entity (event). The approach used by the CMS DAQ is to distribute the event builder over a network of computers to achieve the required performance. Within the XDAQ framework, event builders are constructed from a combination of three communicating applications, a readout unit (RU), a builder unit (BU) and an event manager (EvM). The RU's buffer input from the data sources, and upon receipt of a control message from the EvM send the next data fragment to the required builder unit. BU's collects all fragments belonging to a single event and output the completed events for further processing and/or storage.

The EvM controls the event building process by mediating control messages between RUs and BUs. The event building protocol is designed such that an arbitrary number of RUs and BUs can be defined to fit any required host and network configurations. The event builder applications benefit from the network and protocol transparency provided by the XDAQ framework, and are therefore re-usable over different network mediums.

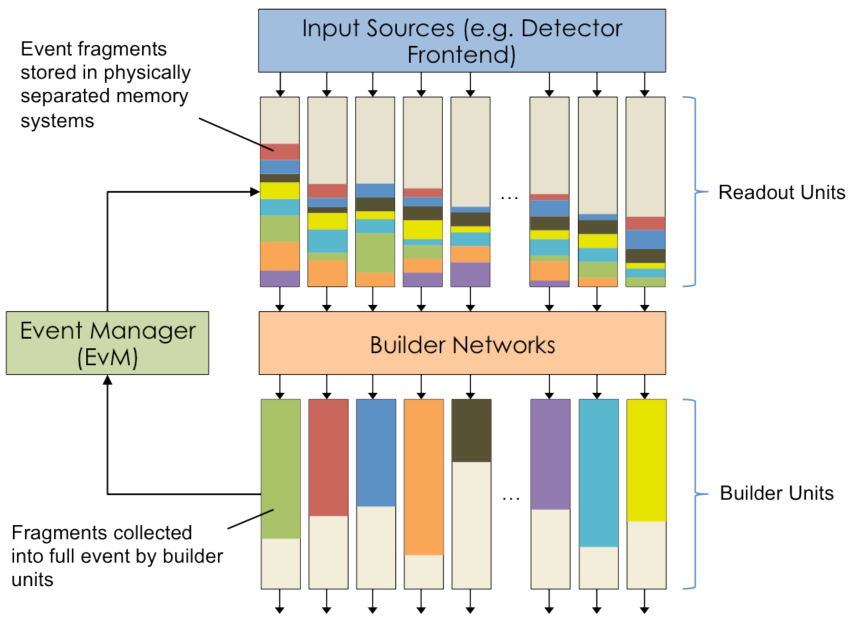
Input Sources (e.g. Detector Frontend)
Event fragments stored in physically separated memory systems
Readout Units
Event Manager (EvM)
Builder Networks
Builder Units
Fragments collected into full event by builder units

Figure 3 – The architecture of the distributed event builder software used in the CMS DAQ

5.Integrating Infiniband

Integration of Infiniband was achieved by including in the XDAQ framework two Peer Transport plugins (ptuDAPL and ptIBV) that interface the uDAPL and ibverbs respectively. The uDAPL and ibverbs libraries were installed as part of the OFED (Open Fabrics Alliance) distribution (v3.5).

Both the uDAPL and ibverbs support multiple transport modes (e.g. send/receive, RDMA write/read) and connection types (reliable/unreliable). The CMS implementations for both peer transports make use of the send and receive operations with reliable connections. There are several advantages to this approach over RDMA operations in the context of event building. First, there is no need to synchronize the sender and receiver with respect to the memory locations used and the completion of work requests. Second, the semantic of sending and receiving fits well with the buffer loaning scheme defined in the framework by providing a high level of control over memory use to the application, as buffers from the memory pools can be directly posted into the send and receive queues. Separate memory pools are used for sending and receiving, which de-couples the two operations reliance on available resources, allows a more efficient allocation of resources and increases the effective level of parallelism that can be achieved. Each memory pool automatically registers all allocated blocks of memory with the Infiniband Host Channel Adapter, and is configured at run time with the required NUMA policy.

To take advantage of multi-core processors, independent threads were dedicated to sending and receiving operations by using a different completion queue for each type of operation. Additional threads were used for event and error handling, and for dispatching messages to the user application. These threads take advantage of the XDAQ framework by being configured to run on specific processing cores at runtime. Fault tolerant behaviour is implemented in the case of an error on a connection. In such an event, all memory allocated to the connection is recycled into the relevant memory pool and no other connections are affected.

The biggest difference between the two implementations is in the methods used for connection management. The uDAPL API provides an asynchronous connection mechanism that supports IP addressing. Within the ibverbs specification, no explicit connection support is defined and different alternatives have been proposed (e.g. rdma_cm). The ptIBV uses a socket based synchronous connection mechanism based on the connector/acceptor model [13] that takes advantage of the option of using IPoIB. Within a given application, for each required remote destination a new QP is created and a connection request issued. On the receiver, a corresponding QP for receiving messages is created upon an incoming connection request and connection information is exchanged, resulting in an established connection. This approach was selected to remain compatible with the XDAQ standard of IP based addressing and due to its simplicity.

6. Preliminary Performance

To perform initial benchmark evaluation of the ibverbs and the uDAPL, we used a small cluster. The setup consisted of 5 nodes of DELL PowerEdge C3220 with dual socket 8-core Intel Xeon E5-2670 processors and 16GB of memory per socket. The operating system running on the nodes was Scientific Linux CERN 6 (SLC6) with the 2.6.32-358.11.1.el6 kernel. Each node was equipped with a Mellanox Connect X-3 VPI Infiniband FDR network card. The uDAPL and ibverbs were installed as part of the OFED v2.x.

The first test was based on a program that measures unidirectional throughput in a N-to-N client-server configuration (see figure 4 left). In this test, N clients send fixed sized messages to N servers using a round robin distribution. The throughput is measured as the number of messages received in a given period for a given message size.

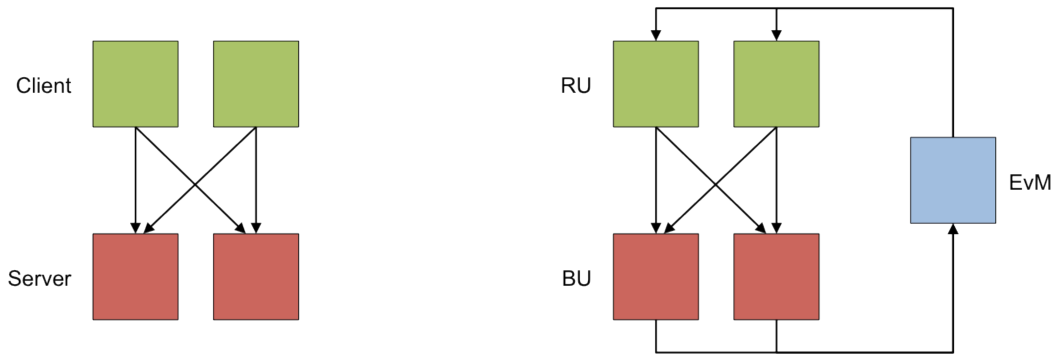

Figure 4 – The setup of the test bed. For N-to-N throughput tests, a 2x2 client/server configuration is used (left). For event building tests (right), an event manager (EvM) is also used which it runs on its own host.

The second test was to measure throughput when adding the additional complexity of event building (figure 4 right), using a simplified version of the CMS event-builder. This test software uses the same protocol as the production event-builder but does not implement some additional features. In this test, event fragments are generated in the RU's at a rate of about 1 MHz. The BU's collect and build events that are then discarded. The flow of data is controlled by the event manager, which takes requests for events from the BUs and issues instructions to the RUs. The measurement is the number of messages received in a given period for a given message size by the BU's.

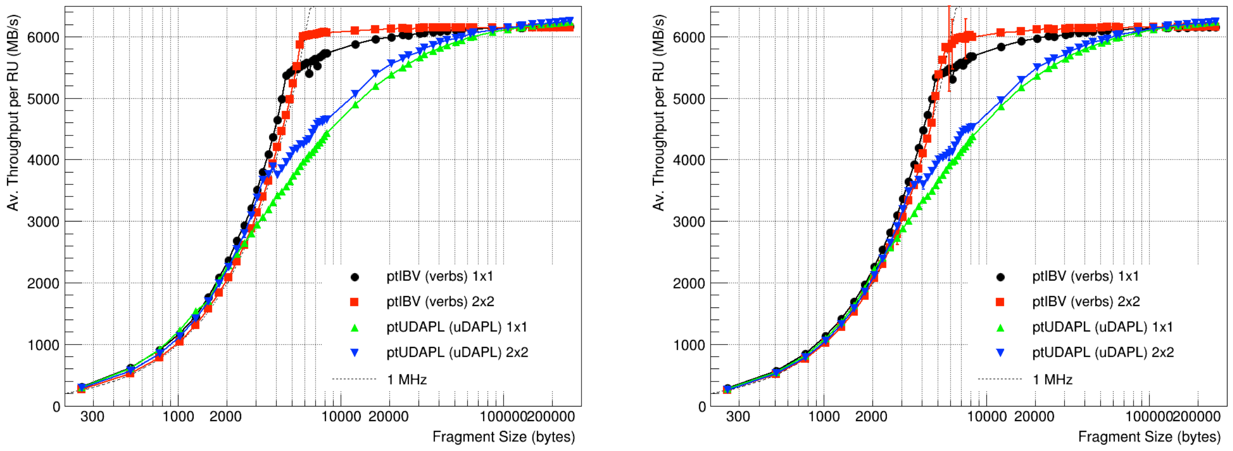

Figure 5 – (Left) N-to-N, (right) event building - Comparisons of the performance for the ptuDAPL and the ptIBV running on the test bed.

Figure 5 shows the results of the initial tests. In all test runs, both peer transports reach saturation at about 6200MB/s (~90% of max theoretical throughput), although the ptIBV performed better for the 4-64kB range. There is no loss in performance when the complexity of event building is added.

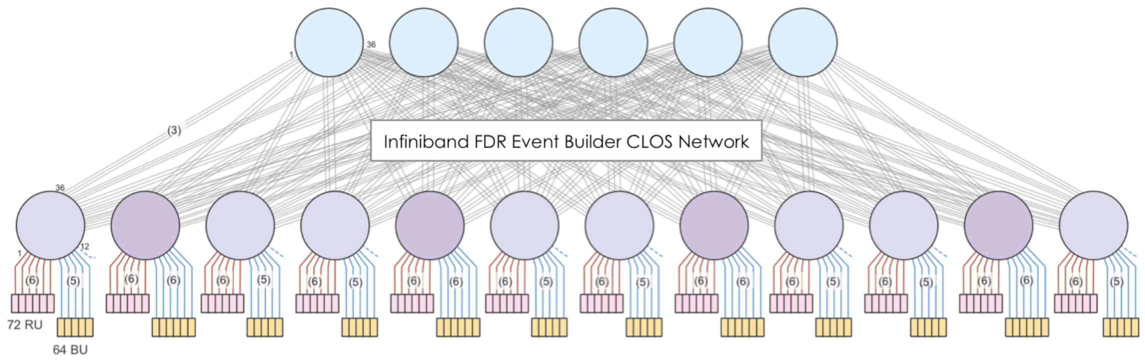

Figure 6 – The CMS DAQ Infiniband FDR CLOS network layout.

The next stage of testing made use of the installed Infiniband FDR CLOS network, shown in figure 6. Based upon the performance demonstrated in the initial tests, scalability tests for the ptIBV were run using the N-to-N benchmarking tool over a range of sizes up to 60x60. Figure 7 (left) shows that for configuration sizes larger than 3x3, the performance does not match that of the smaller tests. However, there is negligible performance loss beyond the initial drop when the setup size increases from 12x12 up to 48x48. When compared against the 100 kHz readout requirement in place for LHC run 2, it is shown that it is possible to reach the necessary throughput in a larger system.

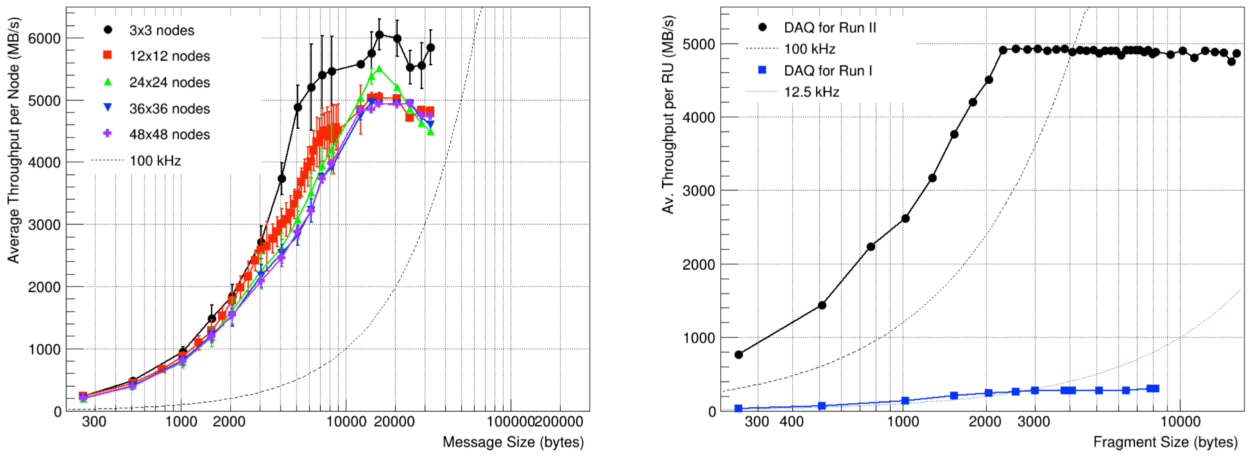

Figure 7 – (Left) Scaling tests using the N-to-N test software. (Right) The average output per Readout Unit used in the CMS DAQ for run 1 (blue) and run 2 (black) along with the relative requirements.

During run 1, the RU received data through a Myrinet NIC using 2x2.4 GB links and output on 3x1 GB Ethernet links. To meet the 100 kHz throughput requirement in run 1, the CMS DAQ was split into eight slices, each with a requirement of 12.5 kHz. When used within the full DAQ system for run 2, the RU will receive data on a 40 GB Ethernet NIC, and output on a 56 GB Infiniband FDR. A comparison of average throughput per RU is shown in figure 7 (right). This plot shows that the RU's that will be used for data acquisition in run 2 are capable of reaching the 100 kHz requirement. In contrast with the DAQ used for run 1, this enables a reduction of physical resources by an order of magnitude.

7. Summary

This paper has shown how Infiniband was integrated into the CMS Online Software framework (XDAQ) to boost event-building performance during data acquisition in LHC run 2. Two new peer transport plugins were developed which provide transparent access to Infiniband networks based upon the uDAPL and ibverbs libraries. By taking advantage of tools within the framework, existing DAQ applications can be augmented without the modification of code to make use of these new plugins. The plugins also take advantage of the event based, zero-copy nature of the libraries they are based upon. The performance that has been demonstrated shows that Infiniband is an interconnect that is ideally suited to facilitate data transmission within a high performance, distributed system such as the CMS DAQ event builder. For the run 2, the ptIBV will be used as the primary Infiniband peer transport in the CMS DAQ based upon the performance compared to the ptuDAPL during preliminary tests.

References

[1] The CMS Collaboration, *CMS Technical Proposal*, CERN LHCC 94-38, 1994

[2] The CMS Collaboration, *The TriDAS Project*, Tech. Des. Rep, 2002, Volume 2: Data Acquisition and High-Level Trigger, CERN LHCC 2002-26

[3] G. Bauer et al, *The new CMS DAQ system for LHC operation after 2014 (DAQ2)*, J. Phys. Conf. Ser. Proceedings of CHEP2013. 513 In Press

[4] Infiniband Trade Association, *(Oct 2004) Infiniband Architecture Specification*, [Online] http://www.inifinibandta.org/specs/

[5] Open Fabrics Alliance, *The Case for Open Source-RDMA and the OpenFabrics Alliance*, [Online], https://www.openfabrics.org/images/docs/LinkedDocs/the_case_for_open_source-rdma_9.6.11.pdf

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

[7] J. Gutleber, S. Murray and L. Orsini, *Towards a homogeneous architecture for high-energy physics data acquisition systems*, Computer Physics Communications, vol. 153, issue 2, pp. 155-163, 2003

[8] The LHC Study Group, *The Large Hadron Collider Conceptual Design Report*, CERN AC 95-05, 1995

[9] E. Hazen et al, *The AMC13XG: a new generation clock/timing/DAQ module for CMS MicroTCA*, Journal of Instrumentation, vol. 8 C12036, 2013

[10] Sutter, H. (2005, March), *The Free Lunch Is Over: A Fundamental Turn Toward Concurrency*, Dr. Dobb's Journal 30 (3) [Online], http://www.gotw.ca/publications/concurrency-ddj.htm

[11] N. Manchanda, K. Anand (2010, May) *Non-Uniform Memory Access (NUMA)*, New York University [Online], http://cs.nyu.edu/~lerner/spring10/projects/NUMA.pdf

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

[13] Douglas C. Schmidt, *Acceptor-Connector: An Object Creational Pattern for Connecting and Initializing Communication Services*, Proceedings of the European Pattern Language of Programs Conference, (1996)