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.

Index Terms—Data acquisition systems, data communication, distributed computing, fast networks, high energy physics computing, software performance.

Manuscript received July 01, 2012; revised June 03, 2013 and September 06, 2013; accepted September 07, 2013. Date of publication November 01, 2013; date of current version December 11, 2013. This work was supported in part by DOE and NSF (USA) and the Marie Curie Program.

H. Racz, H. Sakulin, C. Schwick, and A. Spataru are with CERN, Geneva, Switzerland (e-mail: Andrea.Petrucci@cern.ch).

G. Bauer, C. Paus, O. Raginel, F. Stoeckli, and K. Sumorok are with Massachussetts Institute of Technology, Cambridge, MA 02139 USA.

U. Behrens is with DESY, Hamburg 22607, Germany.

J. Branson, J. Coarasa Perez, A. Holzner, M. Sani are with University of California San Diego, La Jolla, CA 92093 USA.

S. Bukowiec, O. Chaze, J. A. Coarasa Perez, 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, G. Polese, A. Racz, H. Sakulin, C. Schwick, and A. C. Spataru are with CERN, Geneva 1211, Switzerland (e-mail: Andrea.Petrucci@cern.ch).

S. Erhan is with the University of California, Los Angeles, CA 90095 USA.

R. K. Mommesen and V. O’Dell are with FNAL, Batavia, IL 60510 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNS.2013.2282340

TABLE I

| 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   |

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 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 required 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 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: Super-Fragment Builder (SFB) and Readout Builder (RB). The first stage (SFB) is based on clos Myrinet [4] network and it receives $\sim 512$ event fragments coming from the FRLs. The data then goes to one of the Readout Builder slices, where a Readout Unit (RU) gathers it from the SFB, and assembles groups of 8 event fragments into super-fragments.

In each slice there are 80 RUs connected to the Super-Fragment Builders, which send the received data to the Builder Units. The Builder Units (\sim 125 for each slice) build and analyze the full events and forward the selected events to mass

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 Perez, 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. Mommesen, 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

IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 60, NO. 6, DECEMBER 2013 4595
The DAQ is composed of a few thousand hosts [5], [6] and of O(20 k) interdependent applications.

The online applications are based on the XDAQ [7] 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. Using 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 $2 \times 10^{34} \text{cm}^{-2}\text{s}^{-1}$ 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. The main motivations for the upgrade of CMS DAQ system are to accommodate sub-detectors with upgraded off-detector electronics and aging of existing hardware (PCs and networks at least 5 years old).

The upgrade plan for the DAQ system is to replace Super-Fragment builder and Readout 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. A key element for modern network technologies is Remote Direct Memory Access (RDMA) that allows computers in a network to exchange data in main memory without involving the processor, cache, or operating system of either computer. The user Direct Access Programming Library [8] (uDAPL) is a very interesting library that defines a direct access framework for all RDMA-capable transports.

The present paper describes the architecture of XDAQ, and the integration of RDMA-capable transports within the framework by means of uDAPL. This reports 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 [9], designed according to the object-oriented model and implemented using the C++ programming language [10]. 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 ap-
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.

The general programming model follows the event driven processing scheme [11] 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 [12]. 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 i2O specification [13] used for efficient and high performance data transmission, and the other on SOAP and XML [14] 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.

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. 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).

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 I2O frames. A much more detailed description of I2O protocol and data format can be found in Section II-C.

Various buffers can be chained together to allow arbitrarily sized data (see Fig. 5). 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. 6) are the entity responsible for conducting the actual exchange of information over a network. Peer transports encapsulate a set of network interfaces, allowing a peer to send and receive 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.

XDAQ exposes two different abstractions to send and receive messages: the Application Context Services and the Peer Transport Agent services. The Application Context Services are convenient API for sending and receiving messages for writing network-agnostic applications. A Peer Transport is lower level 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. I2O Protocol and Data Format

The framework supports the exchange of messages using the I2O binary data format. I2O (Intelligent Input Output) is originally a specification for an I/O architecture developed by a consortium of computer companies called the I2O special Interest Group (SIG). I2O is designed to eliminate I/O bottlenecks by utilizing special I/O processors (IOPs). In particular, I2O 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 I2O 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.

I2O 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. 7, encapsulates the originator of the message and destination identifier of the application that shall receive the message. A callback function is invoked upon receipt of a
message by mapping the number indicated in the Function ID field to a defined C++ method. The callback can be one of the standard I2O functions as indicated in the field Function ID, or a user supplied callback, by setting Function ID as 0xFF and providing numeric value for the XFunction field in the extended header. Any user data can be inserted into the message after the extended header.

I2O 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. I2O defines, that the ordering of bytes on the network is Little Endian, aligned to 32 or 64 bit boundaries.

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.

I2O is used as an application level protocol (see Fig. 8). I2O 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-lending 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 arithmetic.

III. DAPL INTEGRATION

In recent years, as network speed has increased toward 100 gigabits per second, CPUs 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, technologies such as Infiniband [15], iWARP [16], and RDMA over Converged Ethernet (RoCE) [17] have been developed that not only allow for a very fast interconnect, but also provide a mechanism known as Remote Direct Memory Access (RDMA) [18] to bypass the operating system and CPU in order to directly move data into application memory.

The OpenFabrics Alliance (OFA) [19] develops, tests, licenses, supports and distributes OpenFabrics Enterprise Distribution (OFED) [20] 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.

As shown Fig. 9, 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) [21] 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. 10, uDAPL defines the API for the kernel level when uDAPL Provider is within OS and below. Each Interface
Fig. 11. ptuDAPL integration with XDAQ framework.

Adapter is controlled by exactly one uDAPL Provider and each uDAPL Provider can control multiple Interface Adapters. There can be multiple DAT Providers controlling disjoint sets of Interface Adapters on a host.

A new peer transport (ptuDAPL) has been implemented for DAT library using I₃O messaging protocol and based on DAT Specification 2.0 [22].

It uses a smart memory pool based on the uDAPL memory region allocator, and the random access to memory with no intermediate management is performed using cookies. The cookie keeps the memory reference, and when uDAPL acknowledges the DAT completion event for a message, the ptuDAPL can free the correct memory reference using the cookie inside the message. 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. 11, 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 the test used a small cluster. The setup consisted of 8 nodes of DELL PowerEdge R710 with dual sockets Intel Xeon E5530 4-core at 2.27 GHz and 3 GB 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 4× Quad Data Rate (QDR) connections with data rate of 32 Gbps (Qlogic HCA, qle7340 4× QDR PCIe), and iWARP adapter at 10 GbE (Chelsio T420-CR 10GBase-SFP RNIC). Each node was connected with an Infiniband switch (Qlogic 12300-BS01-4× QDR) and 10 GbE switch (Voltaire Vantage 6048).

V. BENCHMARKS

To evaluate the different network technologies three different benchmark tests were performed:

- 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

A XDAQ application called roundtrip was developed to measure the latency of sending a package between two nodes. A time packet travels 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. 12).

Fig. 13 shows the latency for sending a package with different fragment size using ptuDAPL over Infiniband (4× QDR) and Ethernet (iWARP). For a packet of 32 Bytes the overhead of XDAQ framework is less than 1 usec for both network technologies compared with the raw measurements.

B. Maximum Throughput Per Node With Stream of I/O Data

In order to calculate the maximum throughput per node a XDAQ application (Multi-Stream I/O) was implemented to send multiple streams I/O data from one source to many destinations. As shown in Fig. 14, throughput per node is measured sending continuously N messages to N receivers and time sampling is done at the receiver’s side.

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. 15. In the
Fig. 14. Diagram of the multi-stream I/O application.

Fig. 15. 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.

Fig. 16. Event builder protocol.

Fig. 17. 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.

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. 15, 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 the test used the CMS RU-Builder [23] 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. 16 shows the event builder protocol. With free capacity available, a BU requests the EVM to allocate 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).

An event builder configuration with an EVM, 3 RUs and 3 BUs has been tested using uDAPL/IB versus TCP/IPoIB (4 x QDR). The throughput per node as a function of fragment size is shown in Fig. 17. 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 $e = 2 - \sqrt{2} \approx 0.59$ for $N \to \infty$ [23]. The performance for TCP/IPoIB is very low compared to uDAPL/IB.

VI. SUMMARY

This paper has 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 the feasibility studies for the CMS event builder, a larger cluster is needed to check the scalability. To do this a new system with 32 nodes of DELL PowerEdge C6220 with dual sockets Xeon E5-2670 8-core at 2.6 GHz and 32 GB of memory is being configured. Each node is equipped with a Mellanox-ConnectX-3 VPI adapter (MCX353A-FCBT) supporting 4× Fourteen Data Rate (FDR) connections with data rate of 54.4 Gbps and 40 GbE. The new setup will be used to perform scalability tests that try to improve the Infiniband efficiency using the Quality of Service and test RoCE technology.

REFERENCES
[1] The CMS Collaboration, CMS Technical Proposal, CERN LHCC 94-38, 1994.
[2] The LHC Study Group, The Large Hadron Collider Conceptual Design Report, CERN AC 95-05, 1995.
[3] The CMS Collaboration, CMS, The TriDAS Project, Data Acquisition and High-Level Trigger, CERN/LHCC 2002-26, 2002, vol. 2, Tech. Des. Rep.
[4] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su, “Myrinet-A gigabit-per-second local area network,” IEEE Micro, vol. 15, no. 11, pp. 29–36, 1995.
[5] G. Antchev et al., “The CMS experiment at LHC,” in Proc. 7th Int. Conf. Advanced Technology and Particle Physics, Como, Italy, Oct. 15–19, 2001.
[6] G. Bauer et al., “CMS DAQ event builder based on gigabit Ethernet,” IEEE Trans. Nucl. Sci., vol. 55, no. 1, pp. 198–202, Feb. 2008.
[7] G. Bauer et al., “The CMS data acquisition system software,” in J. Phys.: Conf. Ser., 2010, vol. 219, p. 022011.
[8] The User-Level Direct Access Transport API (uDAPL) [Online]. Available: http://www.datacollaborative.org/udapl.html
[9] P. Bernstein, “Middleware: A model for distributed systems services,” Commun. ACM, pp. 86–98, Feb. 1996.
[10] J. Gutebrer and L. Orsini, “Software architecture for processing clusters based on I2O,” Cluster Comput., vol. 5, no. 1, pp. 55–64, 2002.
[11] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sizer, M. E. Fieczynski, D. Becker, C. Chambers, and S. J. Eggars, “Extensibility, safety and performance in the spin operating system,” in Proc. 15th ACM Symp. Operating System Principles, Dec. 1995, pp. 267–284.
[12] P. Pardyak and B. N. Bershad, “Dynamic binding for an extensible system,” in Proc. 2nd USENIX Symp. Operating Systems Design and Implementation, 1996, pp. 201–212.
[13] “Intelligent I/O (I2O) Architecture Specification v2.0,” I2O Special Interest Group, 1999.
[14] J. Boyer, March 2001, Canonical XML Version 1.0, W3C [Online]. Available: http://www.w3.org/TR/xml-c14n
[15] “InfiniBand Architecture Specification” InfiniBand Trade Association, Oct. 2004 [Online]. Available: http://www.infinibandta.org/specs/
[16] H. Shah et al., Oct. 2007, Direct Data Placement over Reliable Transports [Online]. Available: http://tools.ietf.org/html/rfc5041
[17] “InfiniBand Architecture Specification Release 1.2.1 Annex A16: RoCE” InfiniBand Trade Association, Apr. 2010 [Online]. Available: http://members.infinibandta.org/kwspub/spec/Annex_RoCE_final.pdf
[18] J. Hilland, P. Culley, J. Pinkerton, and R. Recio, RDMA Protocol Verbs Specification, Apr. 2003 [Online]. Available: http://www.rdmaconsortium.org/home/draft-hilland-wARP-verbs-1.0-RDMA.pdf
[19] OpenFabrics Alliance [Online]. Available: http://www.openfabrics.org/
[20] OpenFabrics Enterprise Distribution (OFED) [Online]. Available: https://www.openfabrics.org/ofed-for-linux-ofed-for-windwos/ofed-overview.html
[21] Direct Access Programming Library (DAPL) [Online]. Available: http://www.datacollaborative.org/
[22] uDAPL API Spec Version 2.0 [Online]. Available: http://www.datacollaborative.org/uDAPL_v20.zip
[23] G. Bauer et al., “The CMS event builder and storage system,” in J. Phys. Conf. Ser. 219:022038, 2010.
[24] M. J. Karol, M. G. Hluchyj, and S. P. Morgan, “Input versus output queuing on a space-division packet switch,” IEEE Trans. Commun., vol. 2, pp. 277–287, 1989.