Available on CMS information server CMS CR -2014/081

The Compact Muon Solenoid Experiment

Conference Report

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

21 May 2014 (v3, 19 June 2014)

Achieving High Performance with TCP over 40GbE on NUMA architectures for CMS Data Acquisition

Tomasz Bawej[2], Ulf Behrens[1], James Branson[4], Olivier Chaze[2], Sergio Cittolin[4], Georgiana-Lavinia Darlea[6], Christian Deldicque[2], Marc Dobson[2], Aymeric Dupont[2], Samim Erhan[3], Andrew Forrest[2], Dominique Gigi[2], Frank Glege[2], Guillelmo Gomez-Ceballos[6], Robert Gomez-Reino[2], Jeroen Hegeman[2], Andre Holzner[4], Lorenzo Masetti[2], Frans Meijers[2], Emilio Meschi[2], Remigius K. Mommsen[5], Srecko Morovic[2,a], Carlos Nunez-Barranco-Fernandez[2], Vivian O'Dell[5], Luciano Orsini[2], Christoph Paus[6], Andrea Petrucci[2], Marco Pieri[4], Attila Racz[2], Hannes Sakulin[2], Christoph Schwick[2], Benjamin Stieger[2], Konstanty Sumorok[6], Jan Veverka[6], Christopher C. Wakefield[2],Petr Zejdl[2]

Abstract

TCP and the socket abstraction have barely changed over the last two decades, but at the network layer there has been a giant leap from a few megabits to 100 gigabits in bandwidth. At the same time CPU architectures have evolved into the multi-core era and applications are expected to make full use of all available resources. Applications in the data acquisition domain based on the standard socket library running in a NUMA Architecture are unable to reach full efficiency and scalability without the software being adequately aware about the CPU, IRQ interrupts and memory affinities. During the first long shutdown of LHC, the CMS DAQ system is going to be upgraded for operation from 2015 onwards and a new software component has been designed and developed in the CMS online framework for transferring data with sockets. This software attempts to wrap the low-level socket library to ease higher-level programming with an API based on an asynchronous event driven model similar to the DAT uDAPL API. It is an event-based application with NUMA optimizations, that allows for a high throughput of data across a large distributed system. This paper describes the architecture, the technologies involved and the performance measurements of the software in the context of the CMS distributed event building.

[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

[a] Now at Institute Rudjer Boskovic, Zagreb, Croatia

Achieving High Performance with TCP over 40GbE on NUMA Architectures for CMS Data Acquisition

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

***Abstract*–TCP and the socket abstraction have barely changed over the last two decades, but at the network layer there has been a giant leap from a few megabits to 100 gigabits in bandwidth. At the same time, CPU architectures have evolved into the multicore era and applications are expected to make full use of all available resources. Applications in the data acquisition domain based on the standard socket library running in a Non-Uniform Memory Access (NUMA) architecture are unable to reach full efficiency and scalability without the software being adequately aware about the IRQ (Interrupt Request), CPU and memory affinities. During the first long shutdown of LHC, the CMS DAQ system is going to be upgraded for operation from 2015 onwards and a new software component has been designed and developed in the CMS online framework for transferring data with sockets. This software attempts to wrap the low-level socket library to ease higher-level programming with an API based on an asynchronous event driven model similar to the DAT uDAPL API. It is an event-based application with NUMA optimizations, that allows for a high throughput of data across a large distributed system. This paper describes the architecture, the technologies involved and the performance measurements of the software in the context of the CMS distributed event building.**

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

I. Introduction

Fundamental changes have been made to processor architectures since the first x86 processor [1] was introduced. As shown in Fig. 1, in the middle of 2000's the processor frequency stabilized and the number of cores per processor started to increase.

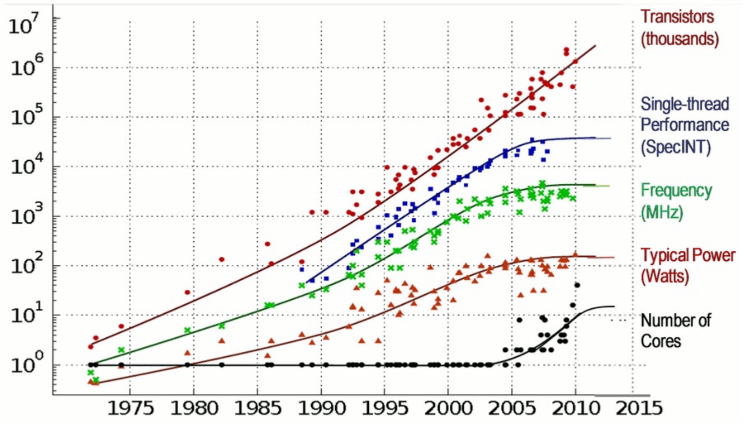


Fig. 1. This chart plots for the last 40 years: the number of transistors per processor (red line), the single-thread performance (blue line), the frequency per core (green line), the typical power consumption (orange line) and the number of cores per processor (black line) – (source: Sam Naffziger, AMD).

With the advent of multi-core architectures and fine grain parallel paradigms the "many-core" era started and software designed with concurrency in mind resulted in more efficient use of new processors [2].

At the same time the network technology has made a giant leap from 10 Mbps to 100 Gbps. Therefore the ability to positively affect application performance involves the selection of adequate switching fabric type and interconnect. In this scope, the two most popular networking solutions in the high performance computing are InfiniBand [3] and Ethernet.

Data Acquisition Systems (DAQ) for High Energy Physics (HEP) experiments use Commodity Off-The-Shelf (COTS) technologies whenever possible. For example, the DAQ system [4] for the Compact Muon Solenoid (CMS) [5] experiment in the first run of Large Hadron Collider (LHC) [6] was built with COTS hardware with the exception of the front-end electronics close to the detector. For LHC run 2 the CMS DAQ system is going to be replaced using state-of-art technologies [7], and the DAQ software needs to be adapted to the new network technologies (Ethernet and Infiniband) and microprocessor architectures.

Manuscript received June 15, 2014.

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

T. Bawej, O. Chaze, C. Deldicque, M. Dobson, A. Dupont, A. Forrest, D. Gigi, F. Glege, R. Gomez-Reino, J. Hegeman, L. Masetti, F. Meijers, E. Meschi, S. Morovic, C. Nunez-Barranco-Fernandez, L. Orsini, A. Petrucci (corresponding author. phone: +41 22 76 70808, fax: +41 22 76 78940, e-mail: Andrea.Petrucci@cern.ch), A. Racz, H. Sakulin, C. Schwick, B. Stieger, C. C. Wakefield and P. Zejdl are with CERN, Geneva, Switzerland.

U. Behrens is with DESY, Hamburg, Germany.

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

G. L. Darlea, G. Gomez-Ceballos, C. Paus, K. Sumorok and J. Veverka are with Massachusetts Institute of Technology, Cambridge, Massachusetts, USA.

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

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

The content of this paper is organized as follows: Section II explains the upgrade of CMS DAQ system for LHC run 2. Section III details the design of the TCP layered architecture. Section IV outlines the tuning for 40 Gbps network using custom TCP settings and processor, memory and IRQ affinities. The experimental results of the TCP layered architecture in the context of the CMS DAQ for run 2 are presented in Section V. A brief conclusion and future work are described in Section VI.

II. The CMS Data Acquisition System for LHC Run 2

The CMS is a general-purpose particle detector designed to study both proton-proton and heavy ion collisions produced at the LHC at CERN in Geneva, Switzerland. In CMS a rejection power of O(10^4) is required in order to reduce the event rate from the 40 MHz LHC beam crossing frequency to an acceptable rate of O(1000) Hz for offline processing and physics analysis. The detector is comprised of about 71 million readout channels. Online event selection is performed using two trigger levels: a hardware based Level-1 Trigger (L1T) and a software-based high-level trigger (HLT). During LHC run 1 (2009-2013) the CMS data acquisition system (DAQ) delivered an excellent performance recording proton-proton collisions at a center-of-mass energy of 7 TeV (2010 and 2011) and at 8 TeV (2012) with 50 ns bunch spacing. The central DAQ availability was more than 99.6 % [8].

In order to reach the energy of 13~14 TeV and a luminosity of $2x10^{34}$ $cm^{-2}s^{-1}$ (LHC run 2) the LHC machine needs to be upgraded during long shutdown 1 (LS1) 2013-2014. Some CMS sub-detector front-end electronics and readout systems will also be upgraded using µTCA-based [9] systems.

The DAQ system must be upgraded in order to cope with the increased instantaneous luminosity requirement for LHC run 2. Furthermore, the DAQ equipment (PCs, networks, etc.) has reached the end of the 5-year replacement cycle. CMS decided to build a new DAQ system that will accommodate sub-detectors with legacy as well as upgraded new off-detector electronics.

A. *Requirements for the CMS DAQ System after LS1*

Table I shows the main parameters of the Trigger and Data Acquisition (TriDAS) system in the case of proton-proton collisions at LHC for run 1 and 2. The important parameters for the capacity of the DAQ are not going to change, in fact the LHC beam crossing will stay to 40 MHz and the CMS Level-1 at 100 kHz. The number of Front End Driver (FED) will be 620 (~500 legacy FEDs; ~120 µTCA FEDs) and the new µTCA FEDs will use Slink-express as the new interface link [9]. The event size will increase to 2 MB and the event builder throughput will be 200 GB/s as a result of the additional detector readout channels and increase of instantaneous luminosity. The high-level trigger (HLT) will be file-system-based to reduce the interdependency of the DAQ system and the HLT. A more detailed description can be found in [10].

B. *Architecture of the CMS DAQ System After LS1*

The architecture of the CMS DAQ system for LHC run 2 is shown in Fig. 2. The system is designed to read out event fragments from around 700 detector Front-Ends Drivers (FEDs) at the level-1 trigger rate of 100 kHz. For the legacy FEDs, data is transferred using SLINK-64 [11] copper links from the sub-detector specific FEDs with SLINK-64 sender cards to a common Frontend Readout Link (FRL) modules.

The FRL is a Compact PCI with two boards connected through a PCI-X interface. The main board provides the interface to the legacy FEDs and the second board is Front-End Readout Optical Link (FEROL). The FEROL [12] has two possible inputs: data coming from the FRL or from the new µTCA FEDs using Slink-Express with a point-to-point optical connection. The FEDs, FRLs and FEROLs are installed in the Underground Service Cavern (USC).

TABLE I
Nominal Parameters of the CMS DAQ

Parameter	LHC RUN I	LHC RUN II	
Beam crossing rate	40 MHz	40 MHz	
Level-1 trigger rate	100 kHz	100 kHz	
Number of front end drivers	640	Legacy 500	µTCA 120
Interface link	Slink64	Slink64	Slink-Express
Fragment size	1-2 kB	1-4 kB	2-8 kB
Event size	1 MB	2 MB	
Event builder throughput	100 GB/s	200 GB/s	

Using a 10 Gbps Ethernet link, FEROLs send data to a Readout Unit (RU) machine with a TCP stream for each FED. Data is transferred from the USC to the Surface Counting room (SC) over a layer of 10/40 Gbps Ethernet switches.

The event building is lossless and performed in two different steps: the FED builder and the RU builder. In the first step a super-fragment is created in the RU from 12 or 16 TCP streams coming from FEROLs. The second part the of event builder system assembles super-fragments into complete events in the Builder Unit (BU) machine. The Event Manager (EVM) supervises the data flow in the RU Builder and receives a data record from the Level-1 trigger via a dedicated FED Builder. The EVM allocates events on request to a BU, which subsequently collects the super-fragments from all RUs. The connectivity between EVM, RUs and BUs is based on an InfiniBand FDR CLOS network.

The file-system-based HLT farm is composed of “BU-FU appliances”. One BU and a fixed number of Filter Units (FUs) are dedicated for each “BU-FU appliance”. The filter farm applications running the physics algorithms read the raw data from the file system located in the BU’s RAM disk, and write the selected events and monitoring meta-data to a local disk. This data is then aggregated over several steps and made available for offline reconstruction and online monitoring. Data between BUs and FUs is exchanged over 1/10/40 Gbps Ethernet switches. Accepted events are transferred over an aggregate bandwidth of 60 Gbps fiber optic connection to the

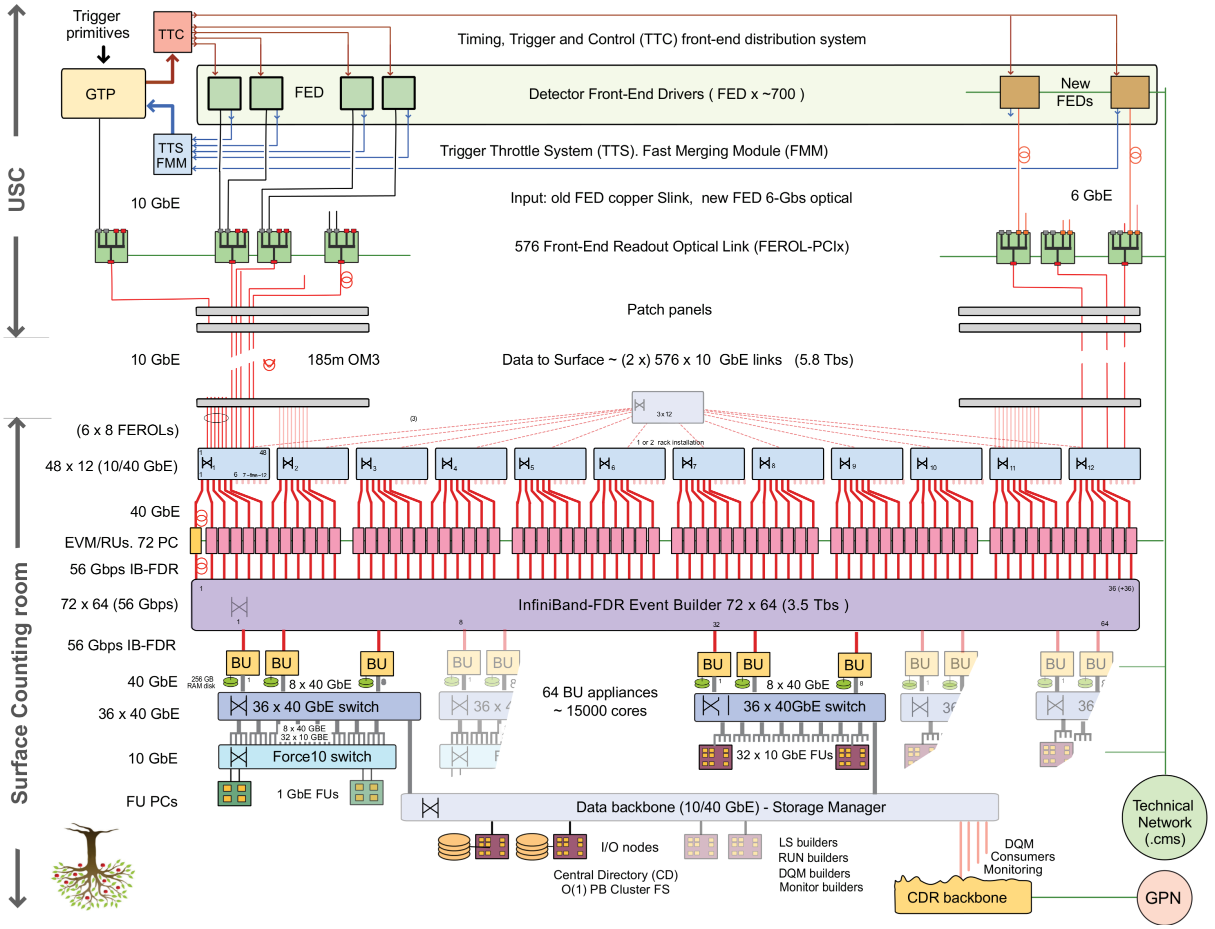


Fig. 2. The diagram shows the CMS data acquisition architecture for LHC run 2. Event data flows from the top to the bottom. The events are built in two stages: super-fragments are built in the Readout Unit (RU) and full events are built in the Builder Unit (BU). The Filter Units (FUs) run the high-level trigger software.

CERN computer center (Tier-0), where they are processed for analysis and archived to a mass storage system.

C. XDAQ Framework

The DAQ applications responsible for the data flow in the CMS DAQ are written using the XDAQ [13] framework. XDAQ is software platform created specifically for the development of distributed data acquisition systems. The development is carried out at CERN for the CMS experiment.

It provides platform independent services, tools for local and remote inter-process communication, configuration and control, as well as technology independent data storage. To achieve these goals, the framework builds upon industrial standards, open protocols and libraries, is designed according to the object-oriented model, and is implemented using the C++ programming language. 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.

A core executive provides the basic functionality, which can be extended at run time with additional binary plugins depending on the requirements, as shown in Fig. 3. Plugins exist for a wide range of additional features, including network communication, memory management and device access. Software for DAQ should be designed to benefit from parallelism available on a hardware/software platform such as multi-core or multi-processor systems. The framework supports three types of access to multithreaded programming: tasks, workloops and timers.

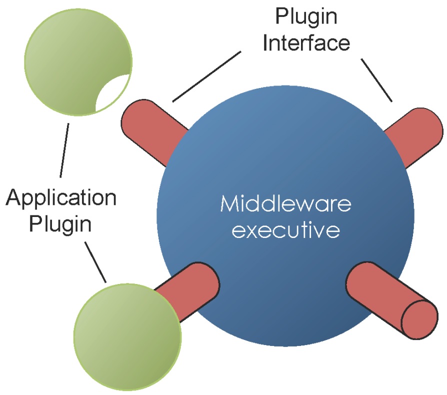


Fig. 3. XDAQ Middleware provides an executive function that supports plug-in modules for additional functionality.

A task is a thin layer over the operating system specific threading API, similar to the Java Task class. Workloops allow for methods to be executed from within a separate thread of control to the invoker thread. A timer uses the task object to schedule a function to be executed at a specific time and/or periodically. In a multiprocessor environment, the framework allows threads to be assigned to run on designated cores, which allows a high granularity of control over where and how processes execute concurrently.

Concerning memory management, XDAQ builds upon the concepts of memory pools and buffer loaning in order to provide an efficient use of memory. 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. With buffer loaning the framework allows applications to deal with data as references to buffers, which can be exchanged between applications with minimal and constant overhead. Such a scheme enables the zero-copy transfer of data through different software layers. When a buffer is no longer needed, the reference to it can be released, and the buffer is returned to the pool and becomes available for re-allocation.

Data transmission between XDAQ processes is carried out through special plug-in components, named peer transports. The peer transport interface relies on the buffer reference abstraction for accessing data allowing the plugin implementation to maintain a zero-copy architecture throughout. By having a peer transport plug-in for each required protocol or network medium, applications using the framework can be protocol and network independent.

III. The TCP Layer Architecture

TCP Layered Architecture (TCPLA) is a lightweight, transport and platform independent user-level library for handling socket processing. TCPLA is modeled against the uDAPL [14] specification, in particular the send/receive semantics it describes. By implementing a wrapper for networking primitives (e.g. sockets), it aims to provide the user with an event driven model of handling network communications, where all calls to send and receive data are performed asynchronously. It supports user defined message formats, and allows the user to control the behavior of the underlying processes. Programming against such an event driven model gives a solid framework to allow well-optimized multi-threaded applications.

The TCPLA represents the various concepts of networking as objects. For example, an Interface Adapter (IA) is an object used to represent a network adapter and an Event Dispatcher (EVD) is an object that queues events for the consumer. These objects are related to one another in an ownership hierarchy. For example, an EVD object is created as the child of a specific IA. Consumers manipulate these objects through handles. Each object type has creation and destruction functions to allocate and de-allocate object resources. The creation functions return a handle with which the consumer can manipulate the object and associate it with other objects.

In TCPLA processes communicate by defining End Points (EPs), which need to be connected to each other before communication can take place. To send or receive data, work requests are posted onto the relative EP's. The completion status of previously posted operations (e.g. work request, connection request) can be checked by using a completion queue mechanism.

The TCPLA library is written in C++ using the XDAQ framework and takes advantage of the memory and thread management facilities that are provided. Fig. 4 shows a UML class diagram of the TCPLA library.

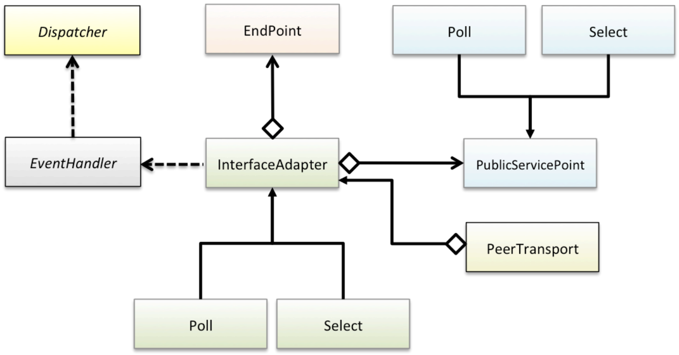


Fig. 4. UML class diagram of object relationships within the TCPLA.

A. TCPLA Design

Communication is achieved using the standard TCP model (socket, bind etc.), but the way data is sent and received from a user's perspective is similar to the uDAPL API. For an optimal usage, understanding how TCPLA relates to TCP is important for performance tuning.

1) TCPLA Event System

TCPLA gives the user an event driven API for communication, connection management and error handling. The user needs to provide an implementation dealing with these events, as what to do can be highly application specific. The event system is the heart of the TCPLA model. Nearly all API invocations are asynchronous in nature, and results are returned in an event.

Completions are logically grouped into event queues, which feed into event dispatchers. Event queue notifications include data transfer completions, connection requests, connection establishment, disconnect notifications, asynchronous errors, and software generated events. Events can be de-queued exactly once. Consumers place operations in queues for processing and either poll or wait on EVD objects for the corresponding events signaling the operation's result. To assist the handling of multiple connections, events contain the information that is necessary for the user to provide in-context responses.

TCPLA uses three queues (in, out and event), as shown in Fig. 5. The in and out queues are filled with buffers that are to be sent or received into. These queues are managed internally, but it is up to the user to provide them with buffers to use. The third queue is the event queue, which is used to schedule events for the 'EventHandler' to consume. Due to the nature of asynchronous communication, event ordering is non-deterministic.

The event handler is the user-derived object that provides implementation for reacting to events. This includes both events relating to normal operation, and events relating to errors and abnormal operation, including peer rejection, loss of connection, timeouts and memory errors.

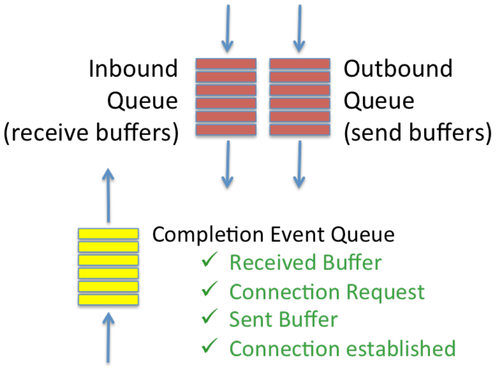


Fig. 5. TCPLA uses three queues to handle events: inbound queue for incoming data, outbound queue for outgoing data and completion event queue for events.

TCPLA allows for two different types of event dispatch queues, waiting and polling. Dispatchers are based on XDAQ workloops and allow the association of threads to different tasks (e.g. receiving completion thread, error event handling thread, send completion thread etc.)

2) *TCPLA Connection*

TCPLA supports reliable connections using a client-server connection model. The client side creates an Endpoint (EP) object and asynchronously submits a connection request to the specified address and port (service point). Upon successful negotiation of a connection, the client receives a 'connection established' event and can begin transmitting data.

Servers handle connection requests using a Public Service Point (PSP). A PSP creates a persistent listener that can service any number of connections. When the socket listen function receives a new connection, a 'connection request' event is dispatched to the user. If the user accepts the connection, an EP is created and a 'connection accepted' event is generated. In turn this triggers the 'connection established' event in the client side and data can be received. The established connection is persistent until either party disconnects or the connection is broken due to error.

All connections are point to point; there is no notion of multicast addressing.

3) *TCPLA Communication*

A user can choose to use select or poll as the underlying mechanism for reading or writing to a socket. To achieve this, TCPLA provides two different versions of InterfaceAdapter and PublicServicePoint, which provide optimized concrete implementations.

When using TCPLA, the receiver must pre-emptively allocate memory for receiving data. Buffers are enqueued by the user, and consumed by the TCPLA when needed. When a buffer is filled, a 'receive complete' event occurs, notifying the user that the buffer is ready to be used. The same model is used for sending data. The enqueued buffer contains the data to be sent, and the return event notifies the user of a send completion. This use of queues allows the send and receive operations to be executed asynchronously in different threads of control.

B. *Developing Peer Transport with TCPLA*

TCPLA provides the building blocks for developing XDAQ peer transports to support different higher-level protocols over TCP/IP capable networks.

The XDAQ distribution specifies three ready to use communication protocols. One is based on the I_2O [15] specification and is used for efficient and high performance data transmission. The second one is based on custom binary protocol and is used for monitoring communication due to its flexibility. The last protocol is based upon SOAP and XML [16] and is used for configuration and control.

Two peer transports have been developed using TCPLA for the CMS DAQ system for run 2: the ptFRL to support the FEROL protocol [11] and the ptUTCP to support the I_2O and B2IN protocols. The ptFRL is designed to manage TCP/IP streams from FEROLs and runs on RU machines. Its main task is to collect and merge data from FEROLs and deliver them to the RU applications. The ptUTCP is the peer transport used when collecting data flow monitoring information in the CMS DAQ system for run 2.

IV. Performance Tuning

Multi-core architectures and fine grained parallel programming paradigms dictate the design of systems and augmenting software to take advantage of available hardware features enabling greater performance.

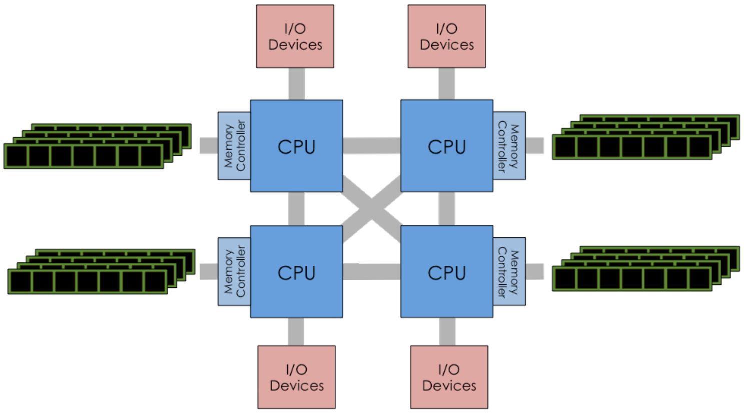


Fig. 6. This diagram shows the different distance between computational core and memory or I/O interrupts in Non-Uniform Memory Access (NUMA) memory design used in multi-core processors.

Within Non-Uniform Memory Access (NUMA) [17] architectures, the distance between processing cores and memory or I/O interrupts varies, with each core having faster access to it's own local memory or interrupt than others (Fig. 6). By properly configuring the affinity for interrupt, memory and processes, it is possible to minimize the access time to memory on multi-core systems.

The default TCP parameters in most Linux distributions are tuned for 100 Mbps or 1 Gbps network interfaces and adjustments should be made when a 40 Gbps network card is used. In the CMS DAQ for LHC run 2, tuning of the TCP settings and the assignment of affinity for memory, CPU's and I/O interrupts are used to achieve maximum performance for the reading of incoming data in the RU's.

By applying the above optimizations a clear difference in performance is observed as shown in Fig. 7.

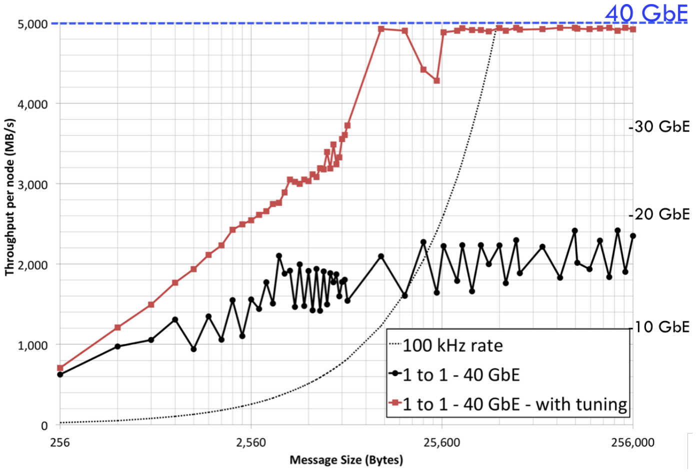


Fig. 7. Throughput in MB/s versus message size in Bytes with point to point application using 40 GbE interface without tuning (black line) and with tuning (red line). The fragment size axis scale is logarithm.

A. TCP Custom Kernel Settings

The most important parameters to change are the kernel TCP socket buffer settings [18]. In TCP the Round Trip Time (RTT) is the time that a packet takes to reach a destination and the acknowledgment packet takes to reach the source. The Bandwidth Delay Product (BDP) is the amount of data that can be in transit at any given time. The BDP is the result of the product of the link bandwidth and the RTT value.

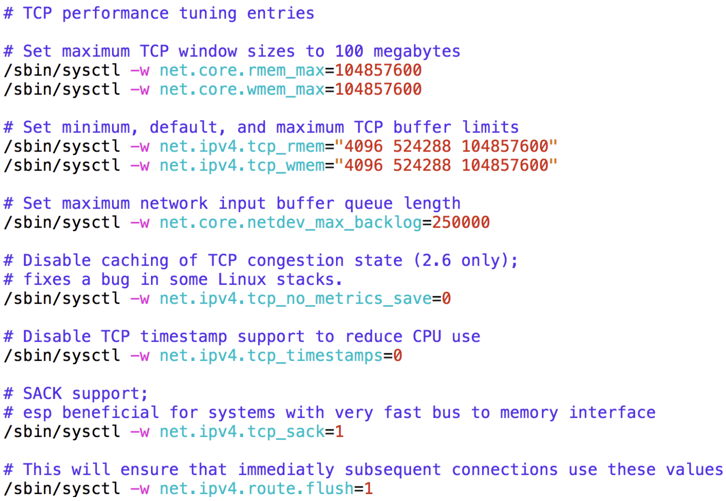
Buffer sizes should be adjusted according to the BDP to allow the maximum number of bytes to be in transit at any given time and prevent traffic throttling.

The Fig. 8 shows the commands used to adjust the TCP parameters to achieve the maximum performance in the RU machines where window sizes, buffer limits, queue length and other TCP parameters are changed.

B. Assigning Affinity with the XDAQ Framework

Assigning affinity for processes and memory allocation helps to reduce the latency when accessing shared data structures and prevents the process scheduler from performing unwanted process migration. The process affinity represents the mapping between a process (thread) and one or a set of processors allowed to run the process. The memory affinity indicates the association of future memory allocations with a single NUMA node.

The XDAQ framework provides configurable allocation and thread policies which are set at runtime according to an XML configuration file. Policies are matched to thread (workloop) or allocator (memory pool) identifiers using regular expressions.



```
# TCP performance tuning entries

# Set maximum TCP window sizes to 100 megabytes
/sbin/sysctl -w net.core.rmem_max=104857600
/sbin/sysctl -w net.core.wmem_max=104857600

# Set minimum, default, and maximum TCP buffer limits
/sbin/sysctl -w net.ipv4.tcp_rmem="4096 524288 104857600"
/sbin/sysctl -w net.ipv4.tcp_wmem="4096 524288 104857600"

# Set maximum network input buffer queue length
/sbin/sysctl -w net.core.netdev_max_backlog=250000

# Disable caching of TCP congestion state (2.6 only);
# fixes a bug in some Linux stacks.
/sbin/sysctl -w net.ipv4.tcp_no_metrics_save=0

# Disable TCP timestamp support to reduce CPU use
/sbin/sysctl -w net.ipv4.tcp_timestamps=0

# SACK support;
# esp beneficial for systems with very fast bus to memory interface
/sbin/sysctl -w net.ipv4.tcp_sack=1

# This will ensure that immediatly subsequent connections use these values
/sbin/sysctl -w net.ipv4.route.flush=1
```

Fig. 8. These commands are used to adjust the TCP parameters in the RU machine for use with 40 Gbps network interfaces.

Within a RU machine, the processor setup is a dual socket 8-core processor with a 16GB NUMA memory node per socket. To take full advantage of the processor architecture, the TCPLA is set to use two dedicated threads for reading sockets, as shown in Fig. 9. These threads are assigned affinities that place them on either side of the core handling the interrupts for the 40 Gbps network card. For the other processor socket, all cores are assigned to run DAQ application tasks with one core kept aside for handling the Infiniband card's interrupts.

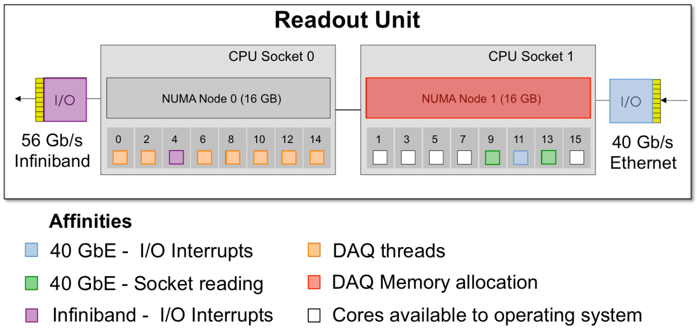


Fig. 9. This diagram shows the I/O interrupt, CPU and memory affinities in the RU Machine.

The memory pools used for DAQ applications allocate buffers on the NUMA node located closest to the 40 Gbps network card. This gives the quickest access time to the threads responsible for reading the sockets, thus reducing the overhead from copying data.

C. *Setting IRQ Affinities*

I/O interrupts are used by I/O devices to notify processors of the completion of an operation. The Advanced Programmable Interrupt Controller (APIC) routes the interrupts to one processor in the system based on the interrupt redirection table. In Linux, it is possible to edit the redirection table using the smp_affinity file of each device in the /proc file system. The irqbalance daemon dynamically changes the table based on the number of interrupts generated for a certain time interval.

The interrupt affinity defines a fixed mapping in the redirection table. In RU machines the interrupt from the 40 Gbps Ethernet card is redirected to a single processor that has local access to the card. To ensure this mapping is fixed, the processors used by DAQ applications and the interrupts from the 40 Gbps Ethernet card have been removed from the resource available to the irqbalance daemon.

V. Preliminary Results

In the RU's, the ptFRL peer transport is responsible to readout fragment data from FEROLs. The goal is to concentrate FED streams in order to optimize the number of RU machines in the system. The following DAQ requirements need to be considered for an L1 trigger operating at 100 kHz:

- 1 FED connected to FRLs/FEROLs with a fragment size between 2 and 4 kB;
- 2 FEDs connected FRLs/FEROLs with a fragment size between 1 and 2 kB.

To perform benchmark evaluation of the new peer transports a DAQ test bed was used.

A. *DAQ Test Bed*

The CMS DAQ group has built a DAQ test bed in order to develop and test software, which allows the testing of various configurations of the DAQ system for LHC run 2. For the purposes of testing the TCPLA, the DAQ test bed consisted of the subset of the CMS DAQ column as required.

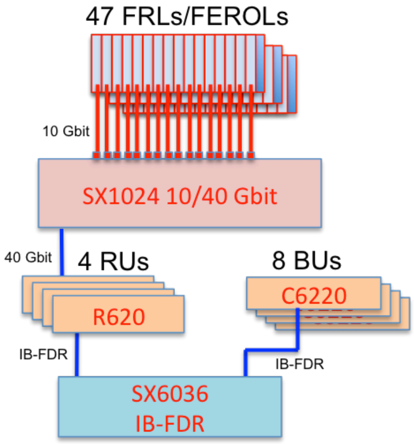


Fig. 10. This diagram shows the DAQ test bed used to evaluate the ptFRL. It composed of 47 FRLs/FEROLs, 4 RUs and 8 BUs. The Mellanox SX 1024 and the Mellanox SX 6036 were used for 10/40 Gbps and 56 Gbps Infiniband interconnections.

The final performance indicator of the ptFRL is based upon the performance achieved while executing simultaneous input and output in the RU. To accommodate this requirement, the tests used the CMS event builder software in emulator mode. FRLs generate the event fragment data (virtual FED) 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. The connection between the RU's and BU's is a 56 Gbps Infiniband network.

The setup consisted of 47 FRLs/FEROLs in 3 crates with 4 RUs and 8 BUs, as shown in Fig. 10. The FEROLs were connected to Mellanox SX 1024 switch using 10 Gbps links. RU nodes are DELL PowerEdge R620's with dual socket Intel Xeon E5-2670 8-core processors at 2.6 GHz and 32GB of memory. Each RU was equipped with two Mellanox ConnectX-3 VPI network cards for 40 GbE and FDR Infiniband connections. DELL PowerEdge C6220's with dual socket Intel Xeon E5-2670 8-core processors at 2.6 GHz and 32GB of memory were used for BU nodes. Each BU had a DELL mezzanine with a Mellanox ConnectX-3 VPI for FDR Infiniband connection. RUs and BUs were connected through a Mellanox SX 1036 and the pause frames were enabled in the switch. The operating system running on the nodes was Scientific Linux CERN 6 (SLC6) with the 2.6.32-279.5.2.el6 kernel.

B. *Performance Measurements*

Preliminary results are presented on throughput measurements for two tests: the first test with one virtual FED per FRL, and the second with two virtual FEDs per FRL. The results are obtained by runs of typically 3 minutes for each measurement.

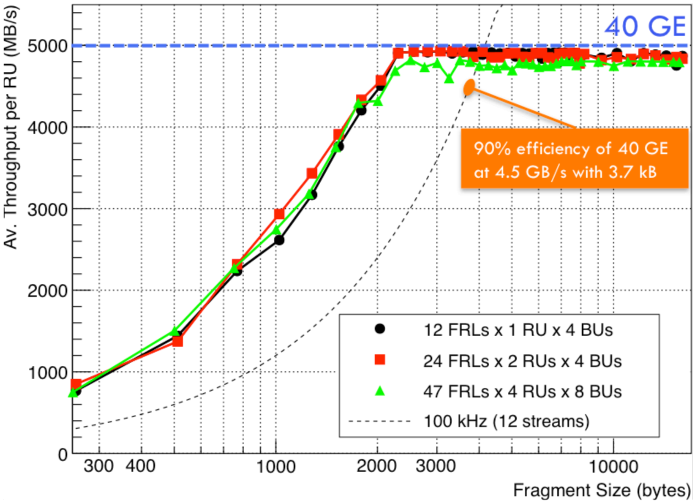


Fig. 11. Incoming throughput per RU in MB/s using event builder software versus fragment size in Bytes for the following configurations: 12 TCP streams from 12 FEROLs, 1 RU and 4 BUs (black line); 24 TCP streams from 24 FEROLs, 2 RUs and 4 BUs (red line); 47 TCP streams from 47 FEROLs, 4 RUs and 8 BUs (green line); 100 kHz for 12 streams L1 trigger requirement for DAQ 2 (dashed line). The fragment size axis scale is logarithm.

For the first test, the incoming throughput per RU as a function of fragment size is shown in Fig. 11 where 12 TCP streams from 12 FEROLs are concentrated in one RU machine. The saturation of the 40 Gbps link (around 5000 MB/s) is reached in all three configurations for fragment sizes above 2.3 kB. When operating at 90% of the 40 Gbps bandwidth with merging from 12 FEDs, the 100 kHz requirement for fragment sizes below 3.7 kB is satisfied.

For the second test, the incoming throughput per RU as a function of fragment size is shown in Fig. 12 where 16 TCP streams from 8 FEROLs are concentrated in one RU machine. Again, the saturation of the 40 Gbps link is reached in all three configurations for fragment sizes above 2.3 kB, and the 100 kHz requirement is met when using 90% of the 40 Gbps bandwidth for fragment sizes less than 2.8 kB.

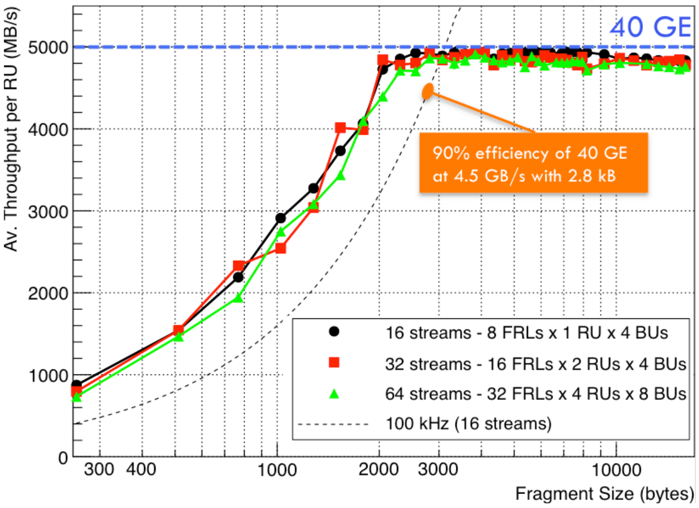


Fig. 12. Incoming throughput per RU in MB/s using event builder software versus fragment size in Bytes for the following configurations: 16 TCP streams from 8 FEROLs, 1 RU and 4 BUs (black line); 32 TCP streams from 16 FEROLs, 2 RUs and 4 BUs (red line); 64 TCP streams from 24 FEROLs, 4 RUs and 8 BUs (green line); 100 kHz for 16 streams L1 trigger requirement for DAQ 2 (dashed line). The fragment size axis scale is logarithm.

VI. Summary

This paper has shown the TCP Layer Architecture approach in the context of the CMS DAQ system for run 2. TCPLA is based on the standard socket library with optimizations for NUMA environments using the XDAQ framework. It has been developed to allow high performance of critical applications in the new CMS DAQ system. This was achieved by exploiting multi-core architectures and optimizing TCP settings. Considerable success has been achieved in modeling the uDAPL API using socket based programming. The XDAQ peer transports that were developed have been shown to operate above the requirements for the LHC run 2, and will be used for both data acquisition and data flow monitoring. Future work will be to demonstrate scalability by expanding the tests over the full DAQ system.

Acknowledgment

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

References

[1] Benj Edwards, (2008, June), “Birth of a Standard: The Intel 8086 Microprocessor” in PC Wold. [Online]. Available http://www.pcworld.com/article/146957/article.html

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

[3] InfiniBand Trade Association. (2004, Oct.). InfiniBand Architecture Specification. [Online]. Available: http://www.infinibandta.org/specs/

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

[5] The CMS Collaboration, “CMS Technical Proposal,” CERN LHCC 94-38, 1994.

[6] The LHC Study Group, “The Large Hadron Collider Conceptual Design Report,” CERN AC 95-05, 1995.

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

[8] G. Bauer et al., “Automating the CMS DAQ,” in J. Phys. Conf. Ser. Proceedings of CHEP2013. 513 In Press.

[9] E. Hazen et al., “The AMC13XG: a new generation clock/timing/DAQ module for CMS MicroTCA,” in J. Instrumentation . Conf. Ser. 8 C12036, 2013

[10] G. Bauer et al., “Prototype of a File-Based High-Level Trigger in CMS,” in J. Phys. Conf. Ser. Proceedings of CHEP2013. 513 In Press.

[11] Attila Racz, Robert McLaren & Erik van der Bij, The S-LINK 64 bit extension specification: S-LINK64, EP Division, CERN

[12] G. Bauer et al., “10 Gbps TCP/IP streams from the FPGA for the CMS DAQ eventbuilder network,” in J. Instrumentation . Conf. Ser. 8 C12039, 2013

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

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

[15] I2O Special Interest Group, Intelligent I/O (I2O) Architecture Specification v2.0 (1999).

[16] J. Boyer. (2001, March). Canonical XML Version 1.0, W3C. [Online]. Available: http://www.w3.org/TR/xml-c14n

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

[18] NASA, TCP Performance Tuning on End Systems. [Online]. Available: http://www.nren.nasa.gov/tcp_tuning.html