Development of an Event Builder software based on InfiniBand network for the LHCb experiment

Matteo Manzali

Department of Mathematics and Computer Science

University of Ferrara

This dissertation is submitted for the degree of

Doctor of Philosophy

Tutor	Co-Tutors
Luca Tomassetti	Francesco Giacomini
	Umberto Marconi

Abstract

The LHCb experiment is one of the four main experiments operating at the Large Hadron Collider at CERN. Its main goals are to make precise measurements of CP violation and to study rare decays of b and c-quark hadrons. The LHCb experiment will undergo a major upgrade during the second long shutdown (2018 - 2019), aiming at collecting an order of magnitude more data the possible with the present detector. The upgraded detector foresees a full software trigger, running at the LHC bunch crossing frequency of 40 MHz. A new high-throughput PCIe Generation 3 based read-out board has been designed on this purpose. The read-out board will allow an efficient and cost-effective implementation of the Data Acquisition System by means of high-speed PC networks.

The aim of this thesis is to study, design and implement an efficient Event Builder (EB) software for the LHCb upgrade. The Event Builder will run on a 500 infrastructure. Each server of the EB relies on two distinct logical components: the Readout Unit (RU) and the Builder Unit (BU). The RU packs the event fragments from the detector, and ship it to a receiving BU in a many-to-one pattern. Each BU gathers the event fragments to assemble them in the full events, which is sent out afterwards for filtering to a specific trigger farm unit. For these reasons I choose InfiniBand as network technology to build up the Event Builder implementation. I developed the EB Communication Layer software, to be used for data transfer among different nodes of the EB exploiting InfiniBand verbs, which is a library that offers an user space API to access the RDMA capabilities of the network device. The InfiniBand network standard is currently the most commonly used interconnection technology in supercomputers. It provides the best performances among the open-standard I/O technologies.

On top of the Communication Layer I created the Logic Layer of the Event Builder, a software that performs the event-building in a realistic way. This software implements a push approach with custom dispatching policies, minimizing the risk of traffic congestion. I also avoided the adoption of a central scheduler, preferring a predefined scheduling strategy. The Event Builder software was tested on different clusters in order to prove its performance and scalability. Finally, I performed preliminary studies on the possibility to use low power architectures for high-throughput data acquisition purposes, using the Event Builder software as use case.

Table of contents

List of figures

List of tables

Introduction

The aim of my PhD thesis is to study, design and implement an efficient Event Builder software for the LHCb upgrade. The LHCb experiment is one of the four major experiments running at the Large Hadron Collider (LHC) at CERN and it will undergo a major upgrade during the second long shutdown (2018 - 2019). This upgrade will concern both the detector and the Data Acquisition (DAQ) system to be rebuilt in order to optimally exploit an expected collision rate of about 40 MHz.

The Event Builder is the component of the DAQ system that receives data incoming from the detector and performs the event composition. Consequently to the increased acquisition frequency foreseen in the upgrade of the experiment, the Event Builder network has to be able to manage a bandwidth of 32 Tb/s with a cardinality of 500 nodes. The first chapter of this thesis deeply describes the LHCb experiment and the readout system foreseen for the upgrade.

In order to achieve these requirements, I decided to rely on the InfiniBand technology for the implementation of the Event Builder network. In the second chapter I compare the most used network technologies in the field of High Performance Computing, describing the reasons that led me to choose InfiniBand. Before taking decisions about the event-building logic, I designed and implemented a software layer that allows to perform connections and communications abstracting the low-level functionalities of InfiniBand. This work is described in the first part of the third chapter.

Then I started to develop an Event Builder software that relies on that software layer to perform the communications. I took several implementation choices to design a proper event-building strategy, including the adopted readout protocol and the related scheduling and dispatching policies. I also developed a software component that simulates the incoming data from the detector with a reasonable data format, which allows to test the Event Builder software in a simulated environment. A detailed description of this work can be found in the second part of the third chapter.

Finally I had the opportunity to run the developed Event Builder on different clusters in order to test the performance and scalability of the software. Moreover, I performed preliminary studies on the possibility to use low power architectures for high-throughput data acquisition purposes, using the Event Builder software as use case. The result of the tests and the testbeds are presented in the fifth chapter.

Chapter 1

The LHCb experiment and its upgrade

In this chapter the reader will find a brief introduction to the LHCb experiment, which is currently taking data at the CERN Large Hadron Collider, and a description of the LHCb detector upgrade plans, foreseen by 2018-19. As a consequence of the upgrade the new detector will yield an order of magnitude more data than the present detector. The improved performance will be obtained operating the event selection by means of an innovative full-software trigger system, and by exploiting a new readout system, which are both entirely based on a high-bandwidth network of commodity PCs.

1.1 CERN

CERN (French acronym for European Organization for Nuclear Research) is the world's largest center for physics research and an unique experimental ground in the field of elementary particle physics. It is located west of Geneva (Switzerland), on the Franco–Swiss border and at the foot of the Jura mountain range (Fig. 1.1), where the geologic and seismic conditions are suitable for a large particle accelerator construction.

The proposal to establish CERN was first sounded by the French physicist and Nobel laureate Louis de Broglie in 1949 at the European Conference on Culture in Lausanne (Switzerland). He proposed the creation of an international organization for fundamental research of such a scale, which would not be feasible by a single national institution alone.

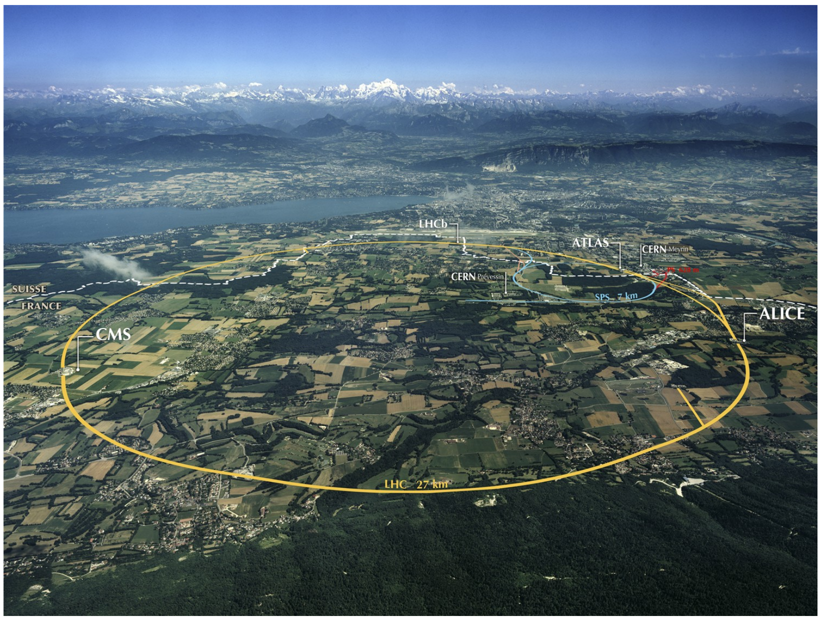


Fig. 1.1 Aerial view of CERN

He was supported by European governments and in 1954 the European Centre for Nuclear Research was born [1]. The initial number of 12 countries which signed the Convention on CERN's membership, today has grown up to 20 member states. About 7000 scientists from approximately 500 research centers and universities in 80 different countries from around the world use CERN as the experimental grounds.

1.2 The Large Hadron Collider

The Large Hadron Collider (LHC) [2], is the particle accelerator with which physicists are able to penetrate as deeply into the matter as never before. Its construction was proposed in 1984 and it was planned that it would operate at a significantly increased, with respect to all previous accelerators, center-of-mass energy of 14 TeV.

The name "Large" comes from the fact that it is the largest and most complicated machine ever built by humanity. In order to save on building costs, the collider was installed in the 27 km underground tunnel at a depth of about 100 meters, which previously hosted the LEP accelerator (Fig. 1.2). The "Hadron" part reflects the fact that the colliding particles are hadrons, i.e. protons and heavy nuclei. Unlike it is predecessor, the LEP project, which

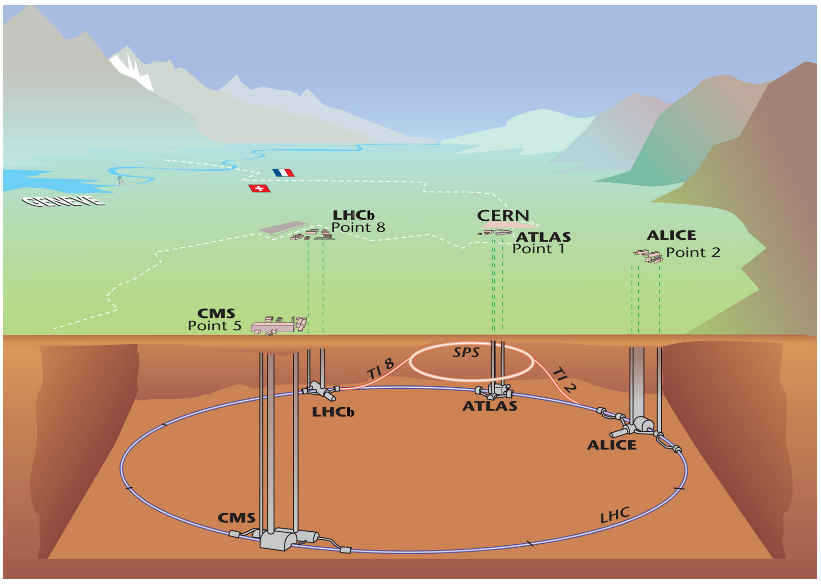


Fig. 1.2 Schematic view of the LHC underground structure

accelerated electrons and positrons, LHC chose to work with proton beams. The reason for that was the aim to reach the maximum possible energy in a given accelerator ring, and the motion of charged particles in a circular orbit is accompanied by energy losses, due to the synchrotron radiation. One can estimate how much higher these losses are during acceleration of electrons in the same ring taking into account that the electron mass is almost 2000 times smaller than the proton's mass, knowing that the intensity of such radiation is inversely proportional to the fourth power of the mass of the accelerated particle. Finally, the "Collider" part of the name points out that it has two beams of particles circulating in opposite directions, which then intersect in special points for particles collisions. In comparison to fixed target experiments, the benefit of such a construction is the higher center-of-mass energy of the colliding particles.

1.2.1 How LHC works

The LHC is indeed a very complex machine. Inside it, the two beams of protons circulate with opposite directions and their speed is close to the speed of light in vacuum. To keep the beam of these energetic particles on a circular track, the accelerator requires magnetic fields which are stronger than at any other accelerator. The technology for generating such magnetic fields, by means of superconducting magnets, has been used before, but there was never an attempt to construct a superconducting apparatus of such a great scale. There are more than 1200 superconducting dipole magnets which are placed along the ring and deflect the particle beams. The maximum value of the magnetic field created by these magnets reaches up to around 8 Tesla. There is yet another category of superconducting magnets: about 400 quadrupole magnets help to optimally focus the two beams along the accelerator ring and in the four collision points where the main detector experiments are located. All of these superconducting magnets operate at a temperature as low as 1.9 K and to cool them down to such a low temperature, nearly 96 tons of superfluid helium circle through the accelerator's ring.

Since the LHC accelerates two beams of particles which move in opposite directions, in reality it consists of two accelerators in one, and before the beams are injected into the main ring, they are accelerated in several sequentially connected machines. Initially, protons reach the energy of 50 MeV in the linear accelerator (LINAC), from which they are rerouted into the Proton Synchrotron Booster (PBS), where they acquire an energy of 1 GeV. Afterwards they travel to the Proton Synchrotron (PS) and Super Proton Synchrotron (SPS) in which they are boosted to energies of 26 GeV and 450 GeV respectively. And only after all these intermediate steps the beams of protons reach the main ring of the LHC, where they are accelerated to the much higher energy of 6,5 TeV per beam, which was achieved in spring of 2015.

1.2.2 The Second Long Shutdown

The LHC has been delivering data to the physics experiments since the first collisions in 2009. After 4 years of data taking, a first Long Shutdown (LS1), which started in February 2013, was triggered by the need to consolidate the magnet interconnections to allow the LHC to operate at the design energy of 14 TeV in the centre-of-mass and has required a 2-years stop [3]. After a second period of operation of more than 3 years, the accelerator complex will be stopped for about 18 months, from July 2018. The main purpose of the Long Shutdown 2 (LS2) is the LHC injectors upgrade. Nevertheless LHC will profit from this period to perform full maintenance of all the equipment, to consolidate part of the machine and to anticipate activities, where possible, of LHC future projects (Fig. 1.3) [4].

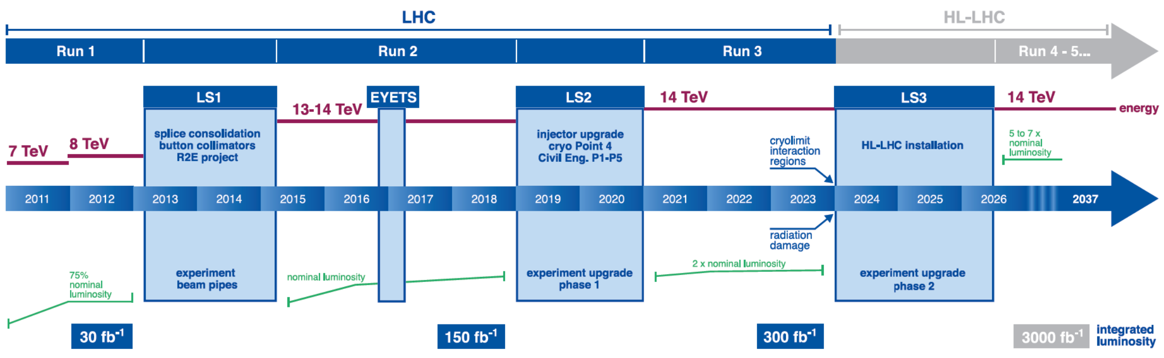


Fig. 1.3 Timeline of the LHC activities

1.2.3 Experiments at the LHC

There are actually seven experiments at the LHC that use detectors to analyse the particles produced by collisions in the accelerator. These experiments are run by collaborations of scientists from institutes all over the world. Each experiment is distinct, and characterized by its detectors. The biggest of these experiments, ATLAS and CMS, use general-purpose detectors to investigate the largest range of physics possible. Having two independently designed detectors is vital for cross-confirmation of any new discoveries made. ALICE and LHCb have detectors specialized for focussing on specific phenomena. These four detectors sit underground in huge caverns on the LHC ring. The smallest experiments on the LHC are TOTEM and LHCf, which focus on "forward particles" – protons or heavy ions that brush past each other rather than meeting head on when the beams collide. TOTEM uses detectors located on either side of the CMS interaction point, while LHCf is made of two detectors which sit along the LHC beamline, at 140 metres either side of the ATLAS collision point. MoEDAL uses detectors deployed near LHCb to search for a hypothetical particle called the magnetic monopole. In the next section an overview of the LHCb experiment will be given.

1.3 The LHCb experiment

The LHCb experiment explores the difference between matter and antimatter. For this it analyses the difference between mesons that contain a b (beauty) quark and antimesons that contain an anti-b quark. B quarks are used because their difference (asymmetry) between quark and anti-quark is larger than with ordinary matter. Just after the big bang, matter and antimatter were evenly distributed. When the universe started to expand, the composition changed such that matter dominated. The LHCb experiment is devoted to the study of the so called CP violation that can account for matter/antimatter asymmetry.

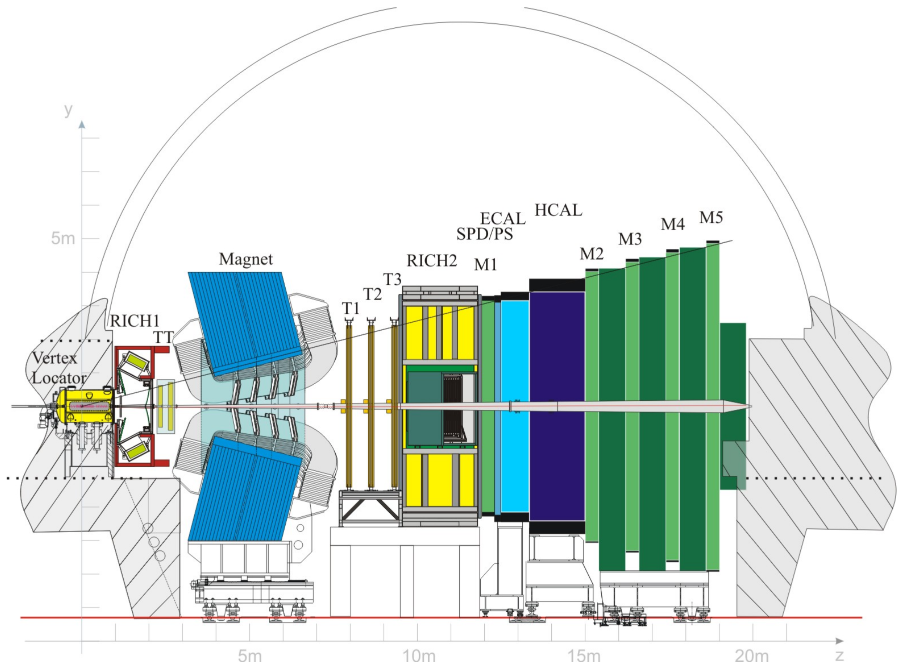


Fig. 1.4 Schematic view of the LHCb detector

The LHCb detector [5] is a forward spectrometer and measures consequently particles moving only in one direction. It consists of many sub detector layers, which are responsible to find the properties and trajectories of particles (Fig. 1.4). The sum of all information allows the particle identification. A very detailed description of the detector can be found on the official LHCb website [6]. The Vertex Locator (VELO) surrounds the collision point and it is responsible for measuring the trajectories of particles close to the interaction point. It allows the separation of primary and secondary vertices. A vertex is the point from where particles emerge, either from a collision (primary vertex) or from the decay of a short lived particle like the B mesons. The Rich-1 detector is located behind the Vertex Locator and allows to identify low-momentum particles. RICH stands for Ring Imaging Cherenkov. Such detectors use the Cherenkov radiation of particles going through a dielectric medium which allows to determine the speed of particles. Rich-1 is followed by the Tracker Turicensis (TT) which is part of the Main Tracker and measures the transverse-momentum of particles. The three Tracking Stations (T1-T3) which are also part of the Main Tracker are located behind the large LHCb dipole magnet. They locate charged particle tracks and can measure their momentum using the bending by the magnetic field. Another Rich detector is located behind the Main Tracker and in contrast to Rich-1, it is responsible to identify particles with high momentum. The next layer consists of the electromagnetic and hadronic calorimeters (ECAL/HCAL), which stop particles and measure their energy. The last layers are the Muon detector stations, that identify Muons, the only charged particles not stopped by the calorimeters. The detection of Muons is important since they are part of many B meson decays.

The detector is specialised for hadron physics and the main goal is to search for CP violation in B meson decays. B physics is term for the study of behaviour of B mesons. B stands for Beauty (quark) and the assumption is that the difference between matter and anti-matter can be understood by evaluating the differences between the beauty and its anti-beauty quark. B mesons consist of a beauty, also known as bottom quark, and either an up-, down-, strange- or a charm-quark. The LHCb detector is designed to record exactly these kind of particles. CP stands for conjugation symmetry parity and it basically defines the difference of behaviour between a particle and its anti-particle. The CP violation is the broken symmetry between them. In 1964, James W. Cronin and Val L. Fitch were the first ones who observed such a violated symmetry in the decay of strange particles (containing a strange quark). They analysed neutral kaons, which were generated at the Alternating Gradient Synchrotron in the Brookhaven Laboratory. They could observe that a certain amount of kaons did not decay like predicted by the CP symmetry. In 1999, the KTEV experiment at Fermilab and the NA48 experiment at CERN could also prove the CP violation of neutral kaons. CP violation can also be observed for B mesons which are heavier than kaons. In 2004, this has been proven for the first time in the BaBar experiment at the Stanford Linear Accelerator Center and Belle experiment at KEK (Japan).

1.4 The LHCb upgrade

One of the main limitations of the current LHCb detector is that the collision rate must be reduced to match the maximum readout rate of 1.1 MHz. The rate reduction is achieved by the Level-0 hardware trigger, which uses the basic events signatures available (calorimeters and muon system objects), operating within a fixed latency of few micro-seconds. Owing to its implementation the Level-0 causes the largest inefficiencies in the entire trigger chain, especially for purely hadronic decays of beauty and charm hadrons. Therefore, one of the main objectives of the LHCb upgrade, planned during the Long Shutdown 2 by 2018-2019, is to remove the hardware trigger bottleneck.

In the foreseen upgraded detector the event yields useful for physics will be maximised by operating a synchronous readout of each bunch-crossing. No more trigger decision is sent to the front-end electronics, making the upgraded LHCb readout completely trigger-free. This requires a change of all the front-end electronics of all the detectors. Also, several detectors will be replaced or upgraded. The upgraded LHCb will operate with a constant instantaneous luminosity five times higher the current value. The luminosity will be kept constant during the fill using the levelling scheme that was successfully operated during Run 1. At this luminosity the expected inelastic collision rate of about 40 MHz, will be processed entirely by the software trigger, which will run on the dedicated Event Filter Farm (EFF). The software trigger selections will be as similar as possible to those applied in offline analyses to maximise trigger efficiencies and to minimise systematic uncertainties.

1.5 The upgraded readout-system design

The main challenge for the trigger-less readout is to build a cost-effective system that can handle the high throughput foreseen by the upgraded detector. As defined in the Letter of Intent (LOI) for the LHCb upgrade [7], the event size will be of the order of 100 KB. At an expected collision rate of 40 MHz the data rate through the Event Builder network will be about 32 Tb/s. The Event Builder described in the Framework TDR [8] is similar to the one used during Run 1 but with a much larger bandwidth. Since the Framework TDR, a new approach has been developed. A new building at the surface will house the core routers and the EFF. This permits utilizing a cost-effective solution in which the readout board, the router and the EFF are located in close proximity. Long distance optical links of 300 meters will be required between the front-end electronics located underground and the readout boards located at the surface. This new design makes it possible to use high-bandwidth cost-effective data-centre link-technology for the Event Builder (Fig. 1.5).

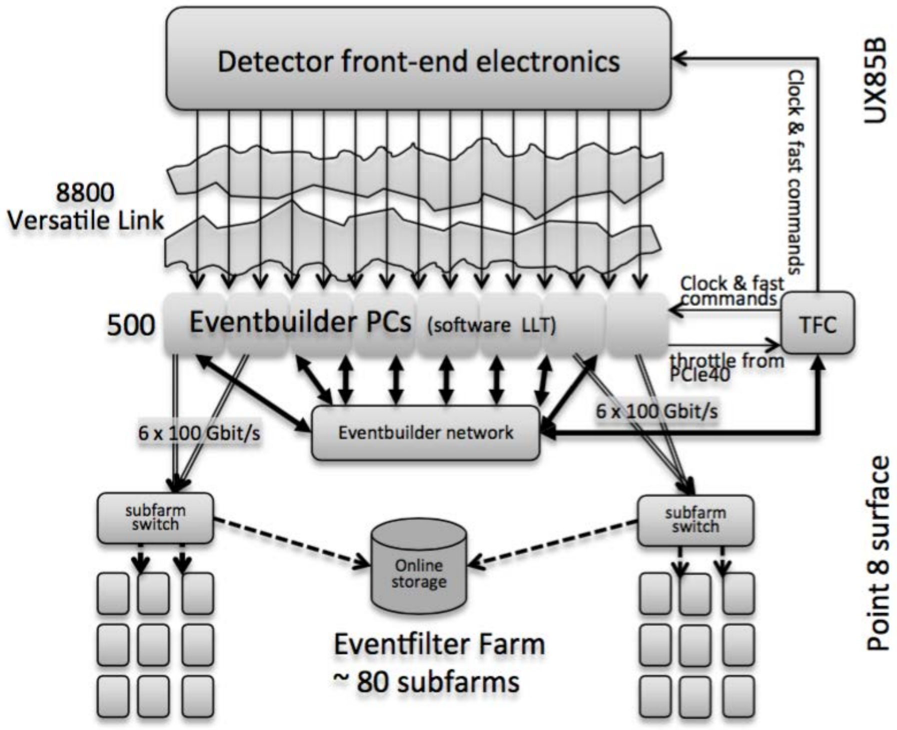


Fig. 1.5 The architecture of the upgraded LHCb readout-system

The central part of the Event Builder in such an architecture consists of dedicated PC servers. These servers interface the front-end electronics via a PCI Express (PCIe) based readout board embedded in each PC server. The input is realised via serial optical links running the GBT protocol [9], while the output is directed to the PC motherboard using the PCI Express protocol (more details can be found in section 1.5.1). All the PC servers involved in the event-building are connected by a large-scale dedicated network. This allows the exchange of event fragments between PC servers, with one of the servers collecting the fragments that belong to the same collision. The PC servers are also connected to the EFF that runs the trigger algorithms. After an extensive R&D program, this architecture was chosen by the collaboration in March 2014 [10].

1.5.1 The readout boards

PCI Express is the high-speed serial computer expansion bus standard designed to replace the older PCI bus [11]. PCIe devices communicate via point-to-point serial links between PCIe ports allowing both to send/receive requests and interrupts. The physical level of the link is composed of one or more lane. Each lane is composed of two differential signaling pairs: one pair for receiving the other for transmitting. Low-speed peripherals use a single-lane link, while, for instance, a graphics adapter board (GPU) typically uses a much wider 16-lane link. PCIe communication is encapsulated in transaction layer packets (TLP) and the Data Link Layer ensure reliable delivery of the TLP between two endpoints via an acknowledgement protocol. Like other high data rate serial interconnect systems, PCIe has a protocol and processing overhead due to the additional CRC and acknowledgements. The nowadays available standard PCIe Gen 3 (PCIe-3) carries a bit rate of 8 Gbit/s per lane, with an overhead of about 2%, due to a 128b/130b encoding scheme.

The readout solution based on PCIe-3 consists of developing the LHCb readout boards as PCIe-3 standard boards, named PCIe40, which act as add-on cards in the motherboards of the servers of the Event Builder. In this approach, represented schematically in Figure 1.6, data from the front-end electronics are transmitted over the GBT-links directly to the event-builder PCs RAM via the PCIe40. Consecutive event fragments transmitted from the front-end electronics are received and buffered at the PCIe40 in Multi Event Packets (MEP) of suitable size and then copied into the event-builder server RAM by means of DMA through PCIe. The proposed PCIe40 is equipped with 24 GBT-links and it is directly connected to the motherboard through 16-lane edge-connector. The PCIe readout requires therefore about 500 PCIe40 cards to read out the whole detector and the same number of Event Builder servers, assuming to have just one PCIe40 card connected to one server [12]. In order to acquire at the desired bandwidth of 32 Tb/s with an Event Builder network composed of 500 nodes, each PCIe40 cards will write at an expected bandwidth of 100 Gb/s.

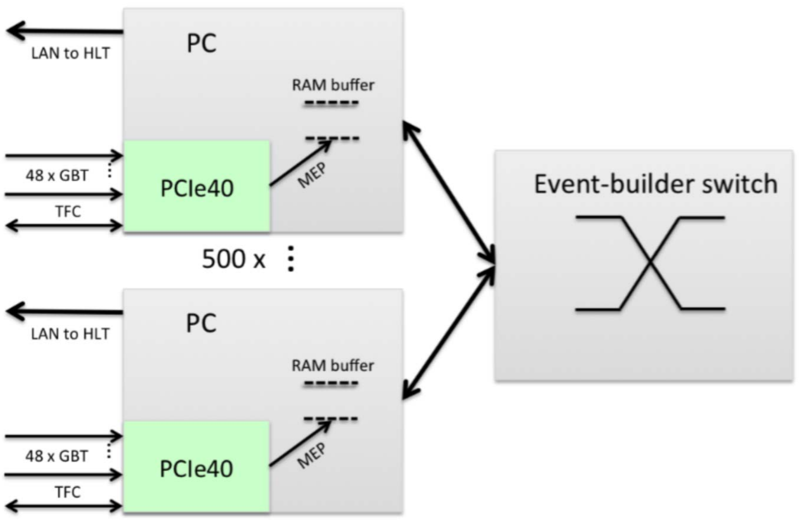


Fig. 1.6 The PCIe based readout system: the PCIe40 readout boards are directly connected to the event-builder PCs through 16-lane PCIe edge-connector

Several studies and tests have proven the compatibility of PCIe-3 technology with the very high performance targets demanded by the future upgrade of the LHCb experiment. This design allows the implementation of a data acquisition system that is scalable, cost-effective and able to pervasively leverage commercial off-the shelves solutions. This strategy allows the development of custom electronics to be kept to a minimum. The PCIe40 will be a common platform for all data acquisition and experiment control tasks within LHCb [13].

Chapter 2

Network Technologies

The event-building requires to bring data from all the readout boards into a single PC server at an estimate bandwidth of about 100 Gb/s, this requires a network able to reach an extremely high bandwidth compared to current standards. Nowadays there are only two network technologies that dominate the High Performance Computing (HPC) market, as can be seen from the TOP500 ranking [14]. The TOP500 project ranks and details the 500 most powerful (non-distributed) computer systems in the world and its chart of interconnect trends shows that the InfiniBand and Ethernet network technologies are widely used compared to the other solutions (Fig. 2.1). In this chapter is explained why InfiniBand is the most suitable candidate as network technology for the event-building. Moreover, it is described how InfiniBand works and its main features.

2.1 Ethernet and InfiniBand roadmaps

At the time of writing both Ethernet and InfiniBand do not reach the bandwith required for the event-building but, according to their roadmaps, there will be versions with an acceptable speed grade in time for the Second Long Shutdown.

The roadmap for Ethernet (Fig. 2.2) is maintained by the Institute of Electrical and Electronic Engineers (IEEE) and is designed to suit the needs of a broad range of applications ranging from home networks to corporate LANs to data center interconnects and even wide

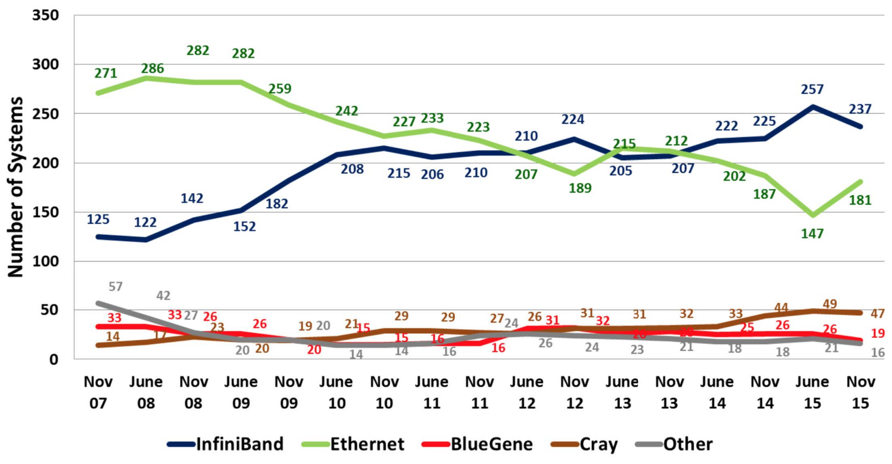


Fig. 2.1 TOP500 Interconnect Trends

area networking. Naturally, each type of application has unique requirements and different speed requirements. For example, client networking does not have the speed requirements that are typical of a data center application. Of this wide range of applications the Ethernet roadmap naturally tends to reflect the bulk of its intended market, even though speed grades more representative of data center needs (40 and 100 GbE) have already been introduced and the 400 GbE is expected in December 2017 [15].

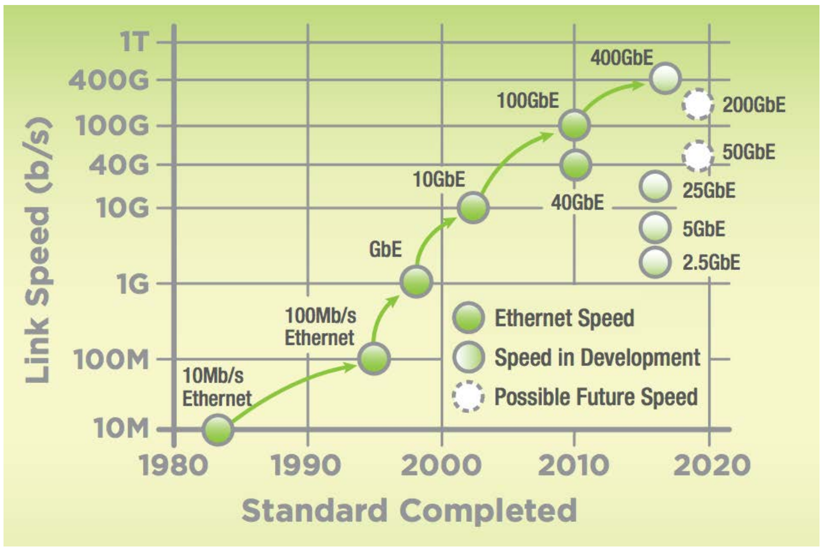


Fig. 2.2 Ethernet Roadmap

The InfiniBand roadmap (Fig. 2.3) is maintained by the InfiniBand Trade Association (IBTA) and has one focus, which is to be the highest performance data center interconnect possible. Commodity InfiniBand components (NICs and switches) at 40 Gb/s and 56 Gb/s have been in wide distribution for years now, the 100 Gb/s speed grade has been introduced last year and the 200 Gb/s speed grade has been announced in the 2017 timeframe [16].

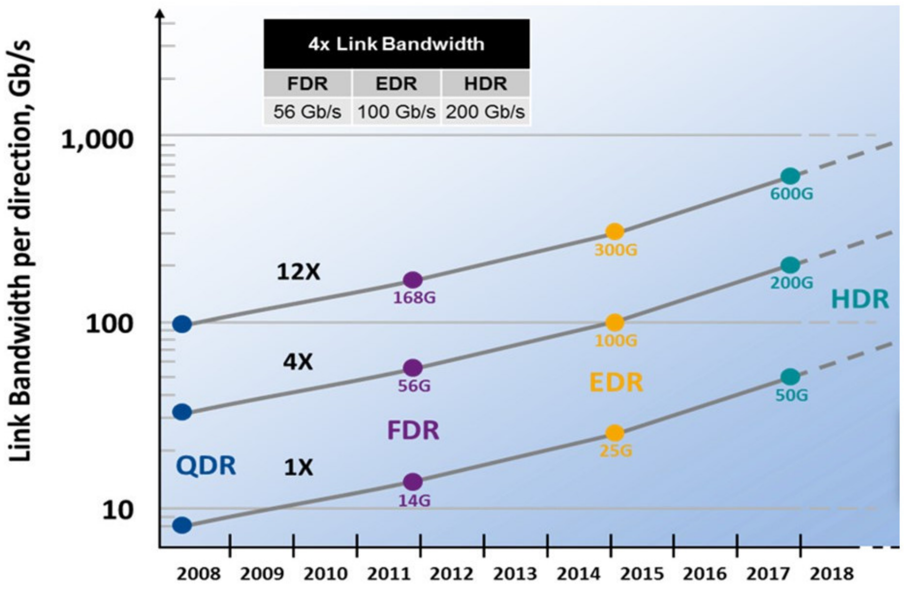


Fig. 2.3 InfiniBand Roadmap

Historically, next generation Ethernet has been deployed first as a backbone (switch-to-switch) technology and eventually trickled down to the end nodes. 10GbE was ratified in 2002, but until 2007 almost all servers connected to the Ethernet fabric using 1 GbE, with 10 GbE reserved for the backbone. The same happened for 40 and 100 GbE: although the specs were ratified by the IEEE in 2010, only from 2014 onwards the first NICs were available. On the other hand, server adapters for InfiniBand are ordinarily available coincident with the next announced speed bump, allowing servers to connect to an InfiniBand network at the very latest speed grades right away. Although the InfiniBand and Ethernet roadmaps are slowly converging, it is still true that Ethernet has a slow adoption curve compared to InfiniBand, making InfiniBand the most suitable candidate as network technology for the event-building.

2.2 Remote Direct Memory Access

The "Remote Direct Memory Access" (RDMA) is the ability to transfer data directly between applications over the network with no operating system involvement and while consuming negligible CPU resources on both sides (zero-copy transfers) [17]. RDMA is quickly becoming a necessity in performance-critical networking: modern HPC interconnects make use of RDMA to achieve throughput approaching hardware bandwidth. The use of RDMA reduces the cost of data movement by eliminating redundant copies throughout the network path and reduces overall resource utilization (Fig. 2.4).

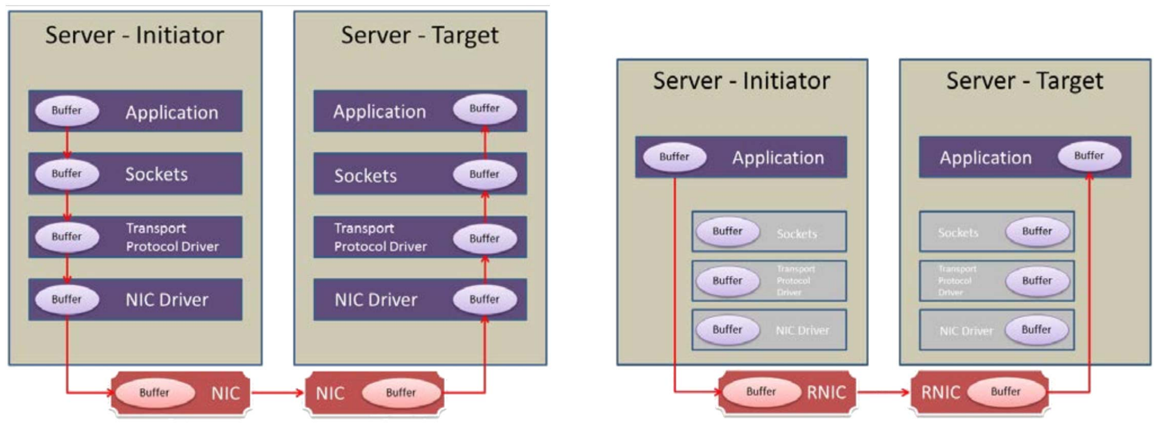


Fig. 2.4 Traditional Interconnect vs RDMA Interconnect

Without the use of RDMA methods, network transfers compete with the local system for the available memory bandwidth. For these transfers, each local copy requires traversing the memory bus twice. Typically, data is received and copied into a device buffer, then copied into an operating system buffer, then finally copied into the application memory. These multiple memory copies are a large bottleneck in HPC systems especially, because memory speeds have not increased at the same rate as CPU and interconnect speeds. For this reason non-RDMA network transfers consume significant amounts of the available memory bandwidth and result in the system CPU(s) stalling on memory accesses.

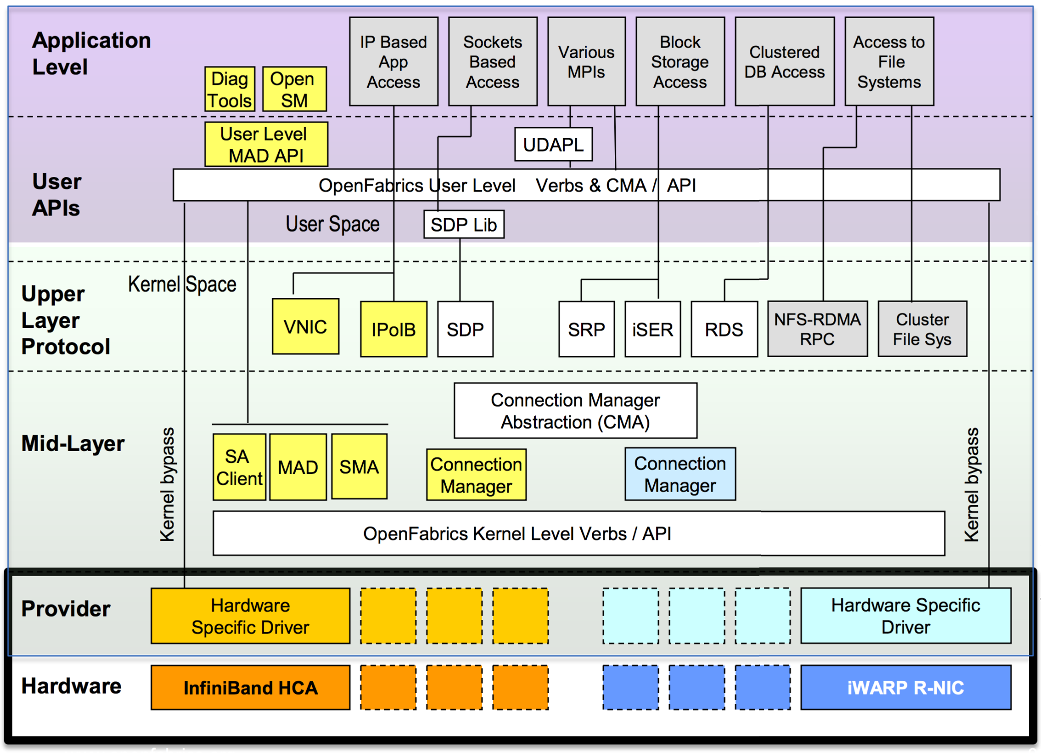


Fig. 2.5 The OFED Stack

2.2.1 A common software for RDMA

The OpenFabrics Alliance (OFA) is an open source-based organization that develops, tests, licenses, supports and distributes the OpenFabrics Enterprise Distribution software (OFED). The vision 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 to run their applications agnostically over every RDMA-aware technology. To that end, the OFA provides tools and development resources to code, refine and publish standards-based, open-source software with the scalability and performance necessary to support mission-critical applications (Fig. 2.5).

The OFED package includes many components to allow the support of RDMA and it is structured in several layers:

- **Application Level**: This layer contains fabric-specifc software such as tools for debugging and diagnostics in an InfiniBand fabric and Open SM, a program that manages and configures an InfiniBand network. In the Application Level there are also various Message Passing Interface (MPI) packages and different high-level applications and access methods for using OFED stack.

- **User APIs**: These are user space APIs that allow development of RDMA enabled applications. This layer also contains the user space API of the RDMA Communication Manager (CMA), a library for setting up RDMA connections.

- **Upper Layer Protocol**: Protocols that facilitate standard data networking, storage and file system applications to operate over InfiniBand. Except for IPoIB, which provides an encapsulation of TCP/IP data streams over InfiniBand, the other upper level protocols (ULPs) transparently provide RDMA and hardware based transport technologies to existing legacy applications.

- **Mid-Layer**: Kernel modules that enables the RDMA support for both user and kernel level.

- **Provider**: Low level driver for the RDMA devices of various vendors such as Mellanox Technologies, Qlogic, Chelsio Communications, IBM and Intel.

The OFED stack offers several ways to build an application that make use of RDMA, such as the ULPs, MPI or uDAPL, but all of these are based on the OpenFabric User/Kernel Level Verbs (or simply "verbs"). The verbs are intentionally not designed as a conventional API. Rather, they are a set of functions and data structures that can be used to build various high-level APIs. They are not oriented toward user applications, because they require knowledge of the internal workings of RDMA hardware and its interactions with the OFED stack. However, any other level of abstraction over verbs may harm the performances, so if the target is to reach the maximum performances allowed by the fabric, the best approach is to develop the application directly on top of verbs.

2.2.2 RDMA-aware technologies

As mentioned above, the work of OFA is not specifically oriented on InfiniBand but more generally on all those technologies that make use of RDMA. Currently, besides InfiniBand, there are two other network technologies that support both RDMA and verbs: RDMA over Converged Ethernet (RoCE) and the internet Wide Area RDMA Protocol (iWARP). Developing an application based on verbs has the considerable advantage of being able to be executed on different network technologies transparently, without being bounded to a single solution.

RoCE

RoCE is a network protocol that allows RDMA operations over an Ethernet network. There exist two RoCE versions, namely RoCE v1 and RoCE v2. RoCE v1 is a link layer protocol and hence allows communication between any two hosts in the same Ethernet broadcast domain. RoCE v2 is an internet layer protocol which means that RoCE v2 packets can be routed. Although the RoCE protocol benefits from the characteristics of a converged Ethernet network, the protocol can also be used on a traditional or non-converged Ethernet network. The RoCE specifications can be found as an annex to the InfiniBand specifications [18] [19].

iWarp

iWarp is an RDMA implementation on top of existing Internet protocols, namely TCP or SCTP on IP, giving it the major advantage of being compatible with the existing Internet infrastructure. iWarp uses both RDMA and OS bypass to move data without the CPU or operating system being involved: in order to facilitate this, a RDMA-capable network adapter is required to handle all network processing, similar to a TCP Offload Engine (TOE). Specifications for the iWarp protocol and software stack exist as IETF RFCs [20]. Currently Chelsio is the only vendor that supports iWarp in its products.

2.3 InfiniBand

InfiniBand (IB) is a high-speed, low latency, low CPU overhead, highly efficient and scalable server and storage interconnect technology. One of the key capabilities of InfiniBand is its support for native RDMA (see section 2.2). InfiniBand efficiency and scalability have made it an optimal performance and cost/performance interconnect solution for the world's leading high-performance computing and storage data centers. InfiniBand is an industry-standard technology and its specifications can be found in the documents "InfiniBand Architecture Specification" [21] [22], which are maintained by the InfiniBand Trade Association.

2.3.1 The InfiniBand architecture

The smallest complete InfiniBand Architecture (IBA) unit is a subnet (Fig. 2.6). A subnet consists of end nodes (e.g. servers), switches, copper or fibre links and a subnet manager. Each end node must have a Host Channel Adapter to set up and maintain the link with the host device. Switches contain more than one InfiniBand port and forwards packets from one port to another to continue the transmission of the packet within a subnet. Subnet management is handled through Software Defined Networking, which controls the network's physical elements and introduces traffic engineering features, often via open and industry-standard interfaces [23].

Links

Links are bidirectional point-to-point communication channels and may be either copper and optical fibre. They can operate at various signaling rates and can then be bundled together to achieve higher throughput: the typical implementation is to aggregate four-link units (4X). The raw signaling rate is coupled with an encoding scheme, resulting in an effective transmission rate. The encoding minimizes the error rate for data sent across the copper or fiber wires and adds some overhead (for example 10 bits transmitted for every 8 bits of data). Currently, InfiniBand systems offer different throughput data rates, as shown in Tab. 2.1.

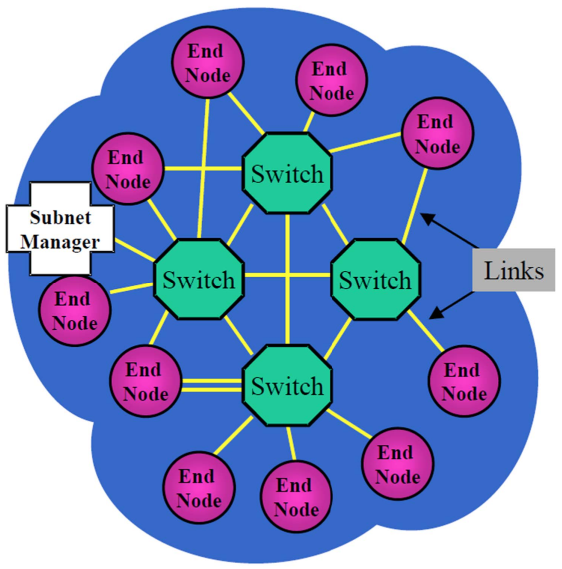


Fig. 2.6 InfiniBand Subnet

Table 2.1 InfiniBand Data Rates

Name	**Acronym**	**Raw Signaling Rate**	**Applied Encoding**	**Aggregated Throughput (4x)**	**Year**
Single Data Rate	SDR	2.5 Gb/s	8b/10b	8 Gb/s	2001
Double Data Rate	DDR	5 Gb/s	8b/10b	16 Gb/s	2005
Quad Data Rate	QDR	10 Gb/s	8b/10b	32 Gb/s	2007
Fourteen Data Rate	FDR	14 Gb/s	64b/66b	54.3 Gb/s	2011
Enhanced Data Rate	EDR	25 Gb/s	64b/66b	96.97 Gb/s	2014
High Data Rate	HDR	50 Gb/s	64b/66b	193.94 Gb/s	2017

Host Channel Adapters (HCA)

The Host Channel Adapter (HCA) is an interface card or controller that bridges the InfiniBand wire and the host system bus. Each end node must have an HCA, which sets up and maintains the link between the host device and the rest of the entities on the network. HCAs provide port connections to other devices. An HCA can be connected to another HCA, to a target device through a Target Channel Adapter (TCA), or to a switch.

Switches

A switch is used to physically connect devices within a network and forward incoming data traffic toward its destination. Switches have multiple ports that process and forward data across cables to the specific device for which it is intended, regulating the flow of traffic within the network. In an InfiniBand fabric, the switch is a crucial piece of the architecture. In fact, InfiniBand is called a "switched fabric" or "switch-based interconnect" because when traffic is forwarded there is a logical connection from one port to another. As more switches are added to the system, there are even more possible paths between devices.

Subnet Managers (SM)

The Subnet Manager (SM) is a software entity that configures its local subnet and ensures its continued operation. It sets up primary and secondary paths between every end point so that traffic flow forwarding decisions are preprogrammed and data arrives in the least amount of time. There must be at least one SM present in a subnet in order to manage all switch and router setups, and to reconfigure the subnet when a link goes down or a new link comes up. The SM can reside on any of the devices within the subnet. All devices in the subnet must contain a dedicated Subnet Management Agent (SMA), which is used by the SM to communicate with the various InfiniBand components. There can be multiple SMs in a subnet, as long as only one is active at any moment. Non-active SMs, known as Standby Subnet Managers, keep copies of the active SM's forwarding information and verify that the active SM is operational. If an active SM goes down, a standby SM takes over responsibilities to ensure that the entire fabric continues with its operation. Software-Defined Networking (SDN) has emerged as a primary means of subnet management, not only controlling the network devices and the data traffic between them, but also adding network flexibility and scalability.

Routers

A router is used to physically connect devices between separate computer networks. When data reaches a router, it reads the address information in the packet and determines the ultimate destination. In Ethernet networking, large subnets are not efficient due to the flooding mechanism and the spanning tree. A router is therefore required to bridge between various smaller subnets. These issues do not exist in InfiniBand, in which large subnets of up to approximately 40,000 nodes can run efficiently. There is no need to divide into smaller subnets, so a router is not a necessity. However, with the ever-growing demands on data centers, InfiniBand router technology is available should there be a need for a subnet of more than 40,000 nodes [24].

Gateways

A gateway is a device within a subnet that serves as a bridge between two protocols. This allows, for example, an InfiniBand cluster to access Ethernet network management or storage interfaces. This allows a company to implement an InfiniBand network with a dedicated interface to existing legacy equipment that runs on other protocols. Gateways are not required to connect InfiniBand clusters with Ethernet networks. Some or all of the compute nodes can have Ethernet access by utilizing Ethernet adapters in addition to their InfiniBand HCAs.

2.3.2 Communication service types

The IBA provides several different types of communication services between endnodes:

- **Unreliable Connection (UC)**: a connection is established between endnodes and messages are sent (transmission is not guaranteed).
- **Reliable Connection (RC)**: a connection is established between endnodes and messages are reliably sent between them.
- **Unreliable Datagram (UD)**: a single packet message can be sent to an endnode without first establishing a connection (transmission is not guaranteed).

- **Reliable Datagram (RD)**: a single packet message can be reliably sent to any endnode without a one-to-one connection.

In the above, "reliably send" means the data is guaranteed to arrive in order, checked for correctness, with its receipt acknowledged. The classes of service, with the exception of the RD one, are strongly reminiscent of similarly-named networking communication services, such as those provided by IP. This is intentional, since they provide essentially the same services. However, these are designed for hardware implementation, as required by a high-performance I/O system. The UC and RD classes are optionally implemented by vendors and currently the RD class is not implemented by any hardware vendor [25].

2.3.3 Queue Pairs and HCA operations

For all the service types, HCAs communicate using special communication channels called Queue Pairs (QP), each of them composed of a Send Queue and a Receive Queue (generically called work queues). A single HCA can own multiple QPs and each QP has a port associated, that is an abstraction of the connection of an HCA to a link (Fig. 2.7). The QP needs to be initialized on both sides of the connection and, in order to allow this, a Communication Manager (CM) component can be used to exchange information about the QP prior to actual QP setup.

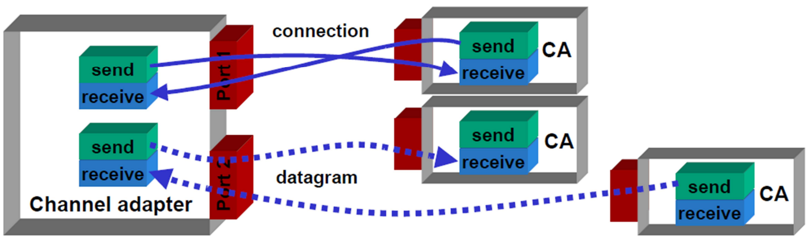


Fig. 2.7 Queue Pairs and ports

Once a QP is established, it can be used to perform RDMA operations such as Send, Receive, RDMA Read, RDMA Write and atomic operations. All of the HCA operations have a similar workflow: each Work Request (WR) is placed on a Send or Receive Queue, it is processed by HCA hardware and consumed, and when processing is completed an entry is optionally placed on the Completion Queue associated (Fig. 2.8).

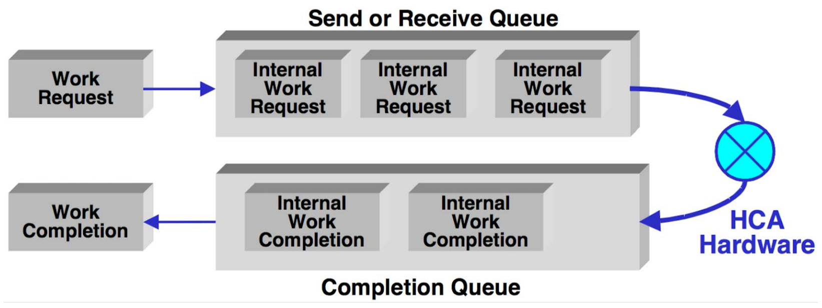


Fig. 2.8 Work Request Processing

SEND / SEND WITH IMMEDIATE

The SEND operation allows to send data to a remote QP's receive queue. The receiver must have previously posted a receive buffer to receive the data. The sender does not have any control over where the data will reside in the remote host. Optionally, an immediate 4 byte value may be transmitted with the data buffer. This immediate value is presented to the receiver as part of the receive notification and is not contained in the data buffer.

RECEIVE

This is the corresponding operation to a SEND. The receiving host is notified that a data buffer has been received, possibly with an inline immediate value. The receiving application is responsible for receive buffer maintenance and posting.

RDMA READ

A section of memory is read from the remote host. The caller specifies the remote virtual address as well as a local memory address to be copied to. Prior to performing RDMA operations, the remote host must provide appropriate permissions to access its memory. Once these permissions are set, RDMA READ operations are conducted with no notification whatsoever to the remote host. For both RDMA READ and RDMA WRITE, the remote side is not aware that this operation being done (other than the preparation of the permissions and resources).

RDMA WRITE / RDMA WRITE WITH IMMEDIATE

Similar to RDMA READ, but the data is written to the remote host. RDMA WRITE operations are performed with no notification to the remote host. RDMA WRITE WITH IMMEDIATE operations, however, notify the remote host of the immediate value.

ATOM FETCH AND ADD / ATOMIC COMPARE AND SWAP

These are atomic extensions to the RDMA operations. The ATOMIC FETCH AND ADD operation atomically increments the value at a specified virtual address by a specified amount. The value prior to being incremented is returned to the caller. The ATOMIC COMPARE AND SWAP will atomically compare the value at a specified virtual address with a specified value and if they are equal, a specified value will be stored at the address.

2.3.4 Memory Model

Control of memory access by and through an HCA is provided by three primary objects: memory regions, memory windows, and protection domains. The relationship among regions, windows, and the underlying virtual memory system is illustrated in Fig. 2.9 and explained in the following.

Memory Regions

Memory regions provide the basic mapping required to operate with virtual addresses. A memory region is created registering a segment of memory with the HCA. This causes the operating system to provide the HCA with the virtual-to-physical mapping of that region and pins the memory (prohibits swapping it out in virtual memory operations). Registration creates an object called L_Key, which must be used with each access to that memory region; it authenticates the use of that region by a QP and specifies access rights. If remote access

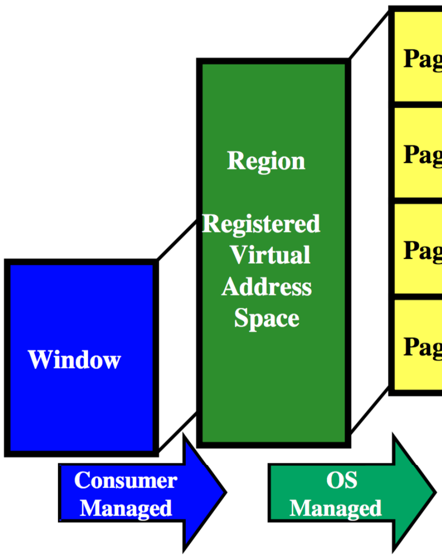


Fig. 2.9 Window - Region - Virtual Memory Relationship

to memory by another endnode is desired for RDMA, a different object called R_Key must be created and used; it must be sent to the endnode performing the memory access and then passed back to the local endnode by the remote one as part of an RDMA request. It is an opaque object that indicates to the HCA which virtual address map must be used and authenticates the requestor.

Memory Windows

Memory registration is necessarily a time-consuming operation: it is likely to be done just once for large segments of memory at a time. Finer-grain and much lower overhead protection is provided by memory windows, so that individual operations can be restricted to refer only to the specific data they would access. A memory window is created within an already-registered region: it specifies a contiguous virtual memory segment with byte granularity. The bounds of a window's segment of memory are re-registered using a work request. Using memory windows is not mandatory but when it is done, subsequent work requests on that work queue must adhere to the memory bounds specified by the window.

Protection Domains

Protection domains effectively glue QPs to memory regions and windows. They are opaque objects, but are expected to correspond roughly to operating system processes or groups of processes which, for example, might share a memory segment. A protection domain is first created and then used when creating a QP, region or window. When applying or using a window or region, the protection domain of the window (region) and the QP using it must match, or an error is raised.

Chapter 3

Design and implementation of the Event Builder

This chapter focuses on the development of the Event Builder software designed for InfiniBand networks and called Large Scale Event Builder (LSEB) [26]. In order to keep separated the communication management from the logic of the event-building, LSEB can be nominally split into two distinct layers, namely the Communication Layer and the Logic Layer. The implementation of the Communication Layer, that relies on the verbs library, and related implementation choices are discussed in sections 3.3 and 3.4. In section 3.2 is introduced the architecture of the Event Builder. Finally in section 3.5 is described the implementation of the Event Builder as the Logic Layer of the LSEB.

3.1 Language, libraries and development environment

Before going through the details of the implementation of LSEB, an introduction to the chosen programming language, libraries and development tools is required. LSEB is developed using the C++ language [27], adopting the C++11 standard and the Boost libraries [28]. C and C++ languages are widely used in High Energy Physics (HEP) and HPC fields for data acquisition and computing purposes, due to the features of flexibility, efficiency, performance, and closeness to the hardware of these languages. In contrast with C, the C++ programming language provides object-oriented functionality keeping a C-like syntax. C++ adds greater typing strength, scoping, and other tools useful in object-oriented programming, and permits generic programming via templates. C++ supports most of C, with a few exceptions such as implicit conversions [29]. The C++ standard template library gives immediate access to very efficient data structures that are not natively present in C and would be complicated to implement. Moreover, using a modern C++ standard, such as C++11, allows the compiler to generate faster and more optimized code. Regarding the development environment, the project makes use of CMake [30] to easily build software and run unit tests in a compiler-independent fashion. The Git versioning system [31] has been adopted to keep track of the software changes and evolution and to facilitate a collaborative software development. A more detailed description of these tools can be found in appendix A.

3.2 The Event Builder architecture

Different dataflow schemas for the upgraded readout-system design have been studied by the LHCb collaboration, analyzing size, complexity and cost in order to establish the most effective one [32]. As introduced in section 1.5, the event-building process is composed by three distinct logical units:

- **The Readout Unit (RU)** acquires a fragment from the detector and sends it to an assigned Builder Unit.
- **The Builder Unit (BU)** collects fragments from all Readout Units which belong to the same collection of bunch-crossings and assembles them into a complete event. It sends complete events to the Filter Unit.
- **The Filter Unit (FU)** receives complete events and selects events for permanent storage.

The core of the Event Builder is formed by all the Readout Units and Builder Units connected with the dedicated network. Moreover, the Filter Units form the Event Filter Farm and are connected to the Event Builder through an additional network.

The chosen dataflow schema, illustrated in Figure 3.1, is called "bi-directional data-flow with combined RU and BU" and has been adopted for the development of LSEB. Such approach foresees combining Readout Units and Builder Units in the same device: for each event a node is selected to be the Builder Unit, it will have one fragment from its own Readout Unit and gets or requests the other fragments from all the other Readout Units to build a complete event. The complete event is then sent to the Event Filter Farm.

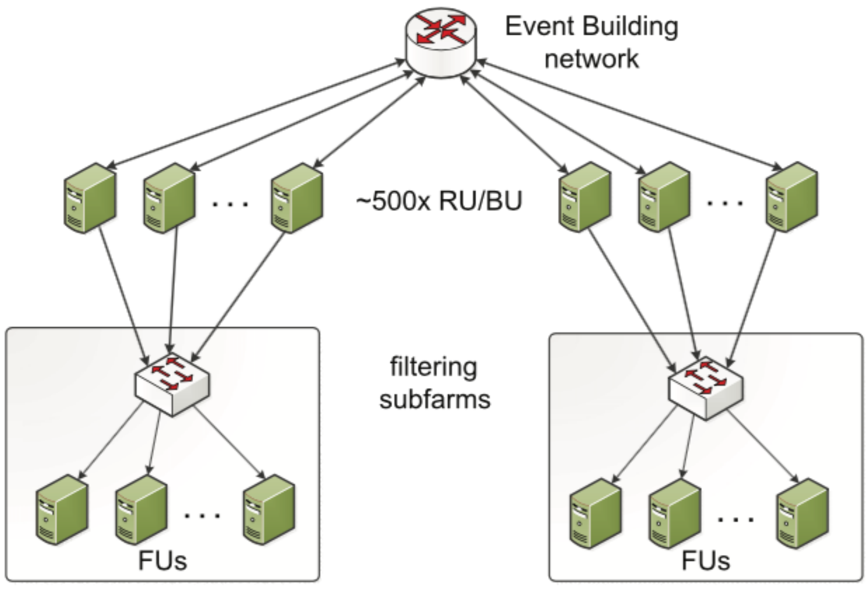


Fig. 3.1 Bi-directional data-flow with combined RU and BU

This dataflow ensures a very powerful separation between the technologies for the fast event-building network and the event-distribution network. The FUs are typically CPU-limited and do not need a high-speed network, so a 10 Gbit/s connection is sufficient for this purpose [32].

3.2.1 The readout protocol

Several studies and simulations have been made by the LHCb community in the last few years in order to identify feasible readout protocols for the upgraded DAQ system [33] [34].

There are mainly two different protocols that depict the communication between Readout Units and Builder Units:

- **Push.** The Push protocol is the simplest solution for data readout and it is used by the LHCb DAQ system today. In this protocol, the Readout Units know wich Builder Unit has been assigned to a specific event and they send packets without waiting for a request from the Builder Units.

- **Pull.** In the Pull protocol the assignment between Builder Units and events is made as soon as the events are ready in the Readout Units. After the assignment has been made, the elected Builder Unit sends a request to the Readout Units for a specific event, then each Readout Unit responds to that Builder Unit with the required event.

The Pull protocol may ensure more flexibility in case of failure of one or more Builder Units but introduces extra latency and complexity due to the data request stage. With the high data rates expected with the upgrade, a multi-stage dispatching protocol could become a critical component and may lead to a performance degradation. On the other hand, the Push protocol allows a linear and simple dataflow but there is the risk that all the Readout Units send fragments to the same Builder Unit almost at the same time, as illustrated in Figure 3.2. This may produce a congestion somewhere in the switch hierarchy or, more likely, the software on the receiving side may not be fast enough to post ready Work Requests on the Receive Queue.

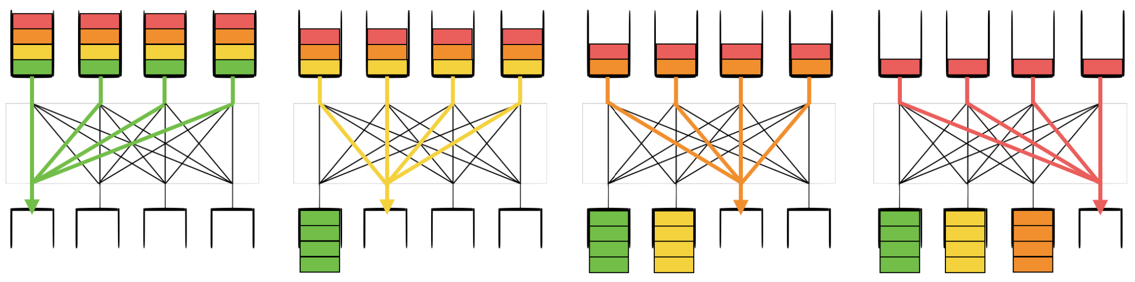


Fig. 3.2 Schematic view of the Push protocol: the Readout Units are concurrently sending fragments to the same Builder Unit

One possible solution to these limits of the Push protocol is to introduce a traffic-shaping like strategy, with the basic idea of having at a certain moment in time all Readout Units sending fragments to mutually exclusive Builder Units. Therefore LSEB implements the Push protocol with a custom dispatching policy: all nodes hosting a Readout Unit - Builder Unit pair have an unique identifier; each Readout Unit can then start to send fragments to the Builder Unit with the subsequent identifier and follows a round-robin order, sending fragments to its local Builder Unit at last, as illustrated in Figure 3.3. The drawback of this approach is that even if a Readout Unit has fragments ready to be sent, it may have to wait in order to respect the adopted dispatching policy, leading to more memory usage respect to the simple Push protocol. However, considering that the average size of an event is about 100 KB and that the PCIe40 readout boards can handle up to 4 GB of memory [35], the needed fragment buffering is still acceptable.

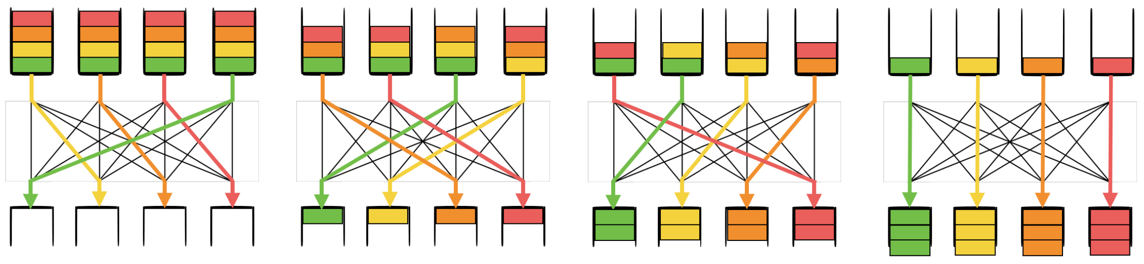


Fig. 3.3 Schematic view of the Push protocol with simple traffic-shaping: the Readout Units are concurrently sending fragments to different Builder Units

3.3 Options for RDMA programming

User-level RDMA programming offers many more options and is more complex than traditional sockets programming, as it requires the programmer to directly manipulate data structures defined by the network interface in order to directly control all aspects of RDMA message transmission. Therefore, the programmer must take decisions which may drastically affect performance. The following sections explain which implementation choices have been taken in order to develop the Communication Layer, a verb based API fitting the Event Builder design.

3.3.1 Completion detection strategy

The basic principle of RDMA is the ability of accessing memory on a remote machine without involving the CPU, leaving it free to perform other tasks. The verbs implement this principle and they are designed to be asynchronous. This means that each transfer call performed with verbs returns immediately instead of waiting for the completion of the requested operation. There are two ways for an application to know about the completion of a work request:

- **Busy polling**. The first completion detection strategy, called "busy polling", is to poll the Completion Queue for a work completion. If it is empty, the function returns the appropriate code for an empty Completion Queue. Otherwise, the function returns the work completion, removing it from the Completion Queue.
- **Event notification**. The second strategy, called "event notification", is to set up a Completion Channel that allows an application to wait until a notification is signaled on this channel. After the notification, the application can obtain the work completion from the Completion Queue.

Repeatedly polling the Completion Queue allows immediate reaction to completions at the cost of full CPU utilization, but requires no operating system intervention. On the other hand, the event notification approach is somewhat less efficient than the polling approach since it introduces the overhead of an interrupt handler and context switches between user and kernel modes [36]. Furthermore, the second strategy forces to adopt an event oriented design for the whole application in order to avoid a busy waiting call somewhere in the logic of the program.

The main aim of the Readout Units in the Event Builder design is to send data to the Builder Units as soon as it is available. When all the send operations are started, the Readout Units cannot do anything but wait for a work completion and, with the high data rates expected, it is likely that the poll operation is almost always successful. Thus, the Communication Layer implements the busy polling strategy, privileging the performance and keeping a linear and simple design.

3.3.2 Communication service type

The verbs allow to establish the connection between two endpoints through the exchange of information that univocally identifies a Queue Pair:

- **LID**. The LID is the "Local Identifier" and it is a unique number given to each port when it becomes active.
- **QPN**. The QPN is the Queue Pair Number and it is the identifier assigned to each queue on the HCA.
- **PSN**. The PSN stands for Packet Sequence Number. In a reliable connection it is used by the HCA to verify that packets are coming in order and that packets are not missing.
- **GID**. The GID is a 128-bit identifier used to identify an endnode, port, switch, or multicast group. It allows routing between different subnets.

This approach forces the developer to implement an out-of-band mechanism in order to collect and share this information before establishing the connections.

An alternative and simpler approach is to establish the connection using the RDMA CM library, that is included in the OFED package. The RDMA CM is a communication manager used to setup reliable connected, unreliable connected and unreliable datagram data transfers. It provides a RDMA transport neutral interface for establishing connections. The API concepts are based on sockets, but adapted for queue pair (QP) based semantics: communication must be over a specific RDMA device, and data transfers are message based. The RDMA CM only provides the communication management (connection setup and teardown) portion of an RDMA API. It works in conjunction with the verbs API for data transfers. The RDMA CM relies on the operating system's network configuration tables to map IP addresses to RDMA devices. This mapping is provided by the IPoIB Upper Layer Protocol, also included in the OFED package, as shown in Figure 3.4.

IPoIB is a protocol that defines how to send IP packets over InfiniBand; the Linux kernel has a driver, called "ib_ipoib", that implements this protocol [37]. The driver creates a network interface for each InfiniBand port on the system, which makes an HCA acts like an

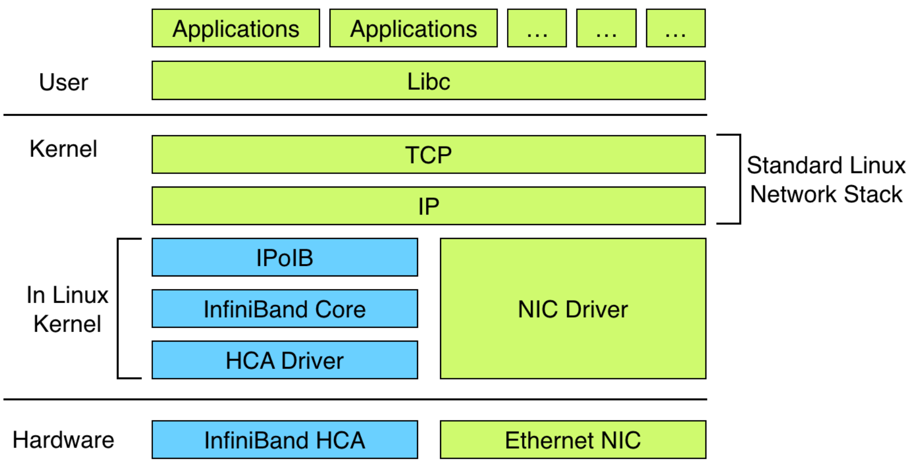


Fig. 3.4 Interfacing between IP-based Ethernet and InfiniBand hardware

ordinary NIC. IPoIB does not make full use of the HCAs capabilities: network traffic goes through the normal IP stack, which means that a system call is required for every message and the host CPU must handle breaking data up into packets. However it means that applications that use normal IP sockets will work on top of the InfiniBand link (although the system is not be able to run the IP stack fast enough to use all the available bandwidth).

Therefore, the Communication Layer relies on the RDMA CM for connection establishment, allowing an easy and socket-like connection approach based on IP addresses.

3.3.3 Work request type

The verbs implement all the work request types foreseen by the InfiniBand Architecture and described in section 2.3.3. Starting from those request types, two different paradigms have been identified for implementing the data transfer logic foreseen by the Event Builder:

- **SEND / RECEIVE**: In the SEND/RECEIVE paradigm a receiver first posts a RECEIVE work request that describes a virtual memory area into which the adapter should place a single message. The sender then posts a SEND work request which refers to a virtual memory area containing the message to be sent. The network adapters transfer data directly from the sender's virtual memory area to the receiver's virtual memory area without any intermediate copies. Since both sides of the transfer are required to post work requests, this is called a "two-sided" transfer. The SEND / RECEIVE is illustrated in Figure 3.5.

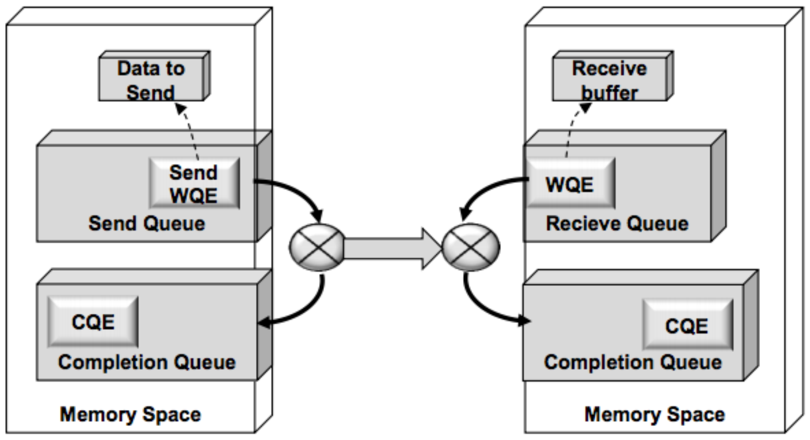


Fig. 3.5 Schema of a SEND / RECEIVE operation

- **RDMA WRITE (WITH IMMEDIATE) / RECEIVE**: The second paradigm is a "one-sided" transfer in which a sender posts a RDMA WRITE request that pushes a message directly into a virtual memory area that the receiving side previously communicated to the sender. The receiving side is completely passive during the transfer. The RDMA WRITE operation is schematically shown Figure 3.6. When the sender needs to notify the receiver, it posts a RDMA WRITE WITH IMMEDIATE request to push a message directly into the receiving side's virtual memory, as for RDMA WRITE, but in this case the work request also includes 4 bytes of immediate (out-of-band) data that is delivered to the receiver on completion of the transfer. The receiving side needs to post a RECEIVE work request to catch these 4 bytes, and the work completion for the RECEIVE indicates the status and the amount of data written with the RDMA WRITE WITH IMMEDIATE operation.

In the Event Builder logic each Builder Unit needs to know the address and the size of each fragment (or collection of fragments) sent by the Readout Units in order to build complete events. With the SEND / RECEIVE paradigm each Builder Unit implicitly obtains such information, being notified of the memory address and the transferred size of each transfer. With the RDMA WRITE (WITH IMMEDIATE) / RECEIVE paradigm the immediate data

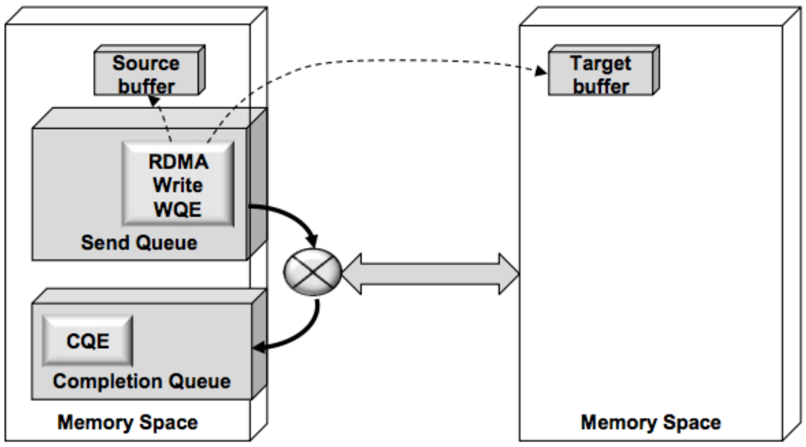


Fig. 3.6 Schema of a RDMA WRITE operation

can be used to communicate the address, but this requires to post a RECEIVE work request by the Builder Unit for each data transfer. Moreover, using this second paradigm requires the Readout Units to be aware of the remote memory in order to write only on those memory areas no longer used by the Builder Units.

Efficiency studies have shown that the performance of these two approaches are essentially equal [38]. However, the use of the first paradigm has the advantage of avoiding the implementation of memory management and synchronization mechanisms. For these reasons the Communication Layer implements the SEND / RECEIVE paradigm to perform RDMA data transfers.

3.3.4 Memory management

The DMA engine responsible for transferring data from main memory to the HCA handles only physical addresses. Thus, the virtual addresses of the communication buffer have to be translated into physical ones and it is also important to ensure that every page of the communication buffer is pinned to prevent swapping. This process is called memory registration.

Registering a Memory Region is a very time-consuming process which requires several operations to be performed by the OS and the driver [39]. In order to avoid registration and deregistration of memory for every transfer, it is preferable to use designated memory buffers, which are registered only once. Furthermore, registering physical contiguous memory can allow the low-level drivers to perform optimizations since lower amount of memory address translations will be required [40]. Thus, it is a good practice to register a large Memory Region and to access to a subset of it each time.

These considerations fit well with the design of the Event Builder, in which a large and contiguous memory area of the Readout Units is written by the readout boards and subsets of it are sent remotely to the Builder Units. Therefore, the Communication Layer allows the registration of a single Memory Region, keeping all the memory contiguous, and avoids temporary registrations/deregistrations. In the Communication Layer the use of the Memory Windows (section 2.3.4) is also avoided, because there is no need of a finer granularity with respect to the Memory Regions.

3.4 The Communication Layer

The Communication Layer is the lowest layer of LSEB and it makes use of verbs to perform RDMA operations and implements all the strategies discussed in the previous section. The Communication Layer exposes a simplified API to the overlying Logic Layer: it handles connections in a TCP like way, using the so called RDMA Connection Manager, and implements one-way communications has foreseen by the Event Builder design. The detailed API of the Communication Layer can be found in the appendix C.1.

3.4.1 Adopted workflow

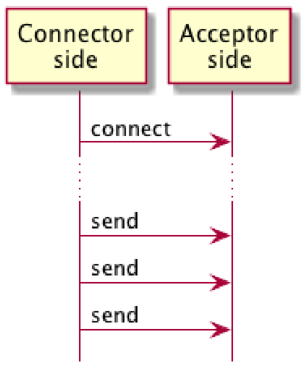
The Communication Layer follows the so called Acceptor-Connector design pattern [41]. The objects that compose the Communication Layer are:

- **Acceptor.** The Acceptor is the object used to listen for incoming connections and to accept them.

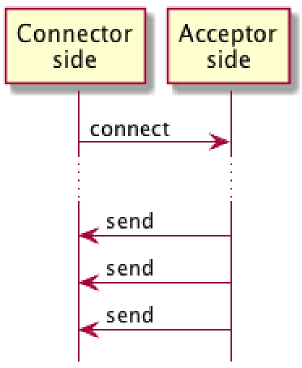
- **Connector.** The Connector is the object used to ask for a new connection.

- **SendSocket.** The SendSocket represents one-way communication channel and it is created by an Acceptor or a Connector when a connection is established. It performs data transfer from the local endpoint to the remote connected one.
- **RecvSocket.** The RecvSocket represents one-way communication channel and it is created by an Acceptor or a Connector when a connection is established. It allows to receive data sent from the remote connected endpoint to the local one.

The Communication Layer foresees two different approaches depending on who performs the send operations between the Acceptor side and the Connector side, as shown in Fig. 3.7.



(a) Send performed by the Connector side



(b) Send performed by the Acceptor side

Fig. 3.7 The different approaches allowed by the Communication Layer

These two approaches follow similar steps, regardless of who performs the send operations. A graphical view of the resulting workflow is shown in Fig. 3.8, assuming that the send operations are performed by the Connector side. The following is a step-by-step description of the workflow of the Communication Layer:

1. The Acceptor initiates a listen for incoming connections. Before performing the real listen, the Acceptor creates a Protection Domain, a Queue Pair with Reliable Connection transport type and the related Completion Queues.
2. The Connector tries to connect to the specified endpoint. Before performing the real connect, the Connector creates a Protection Domain, a Queue Pair with Reliable Connection transport type and the related Completion Queues.

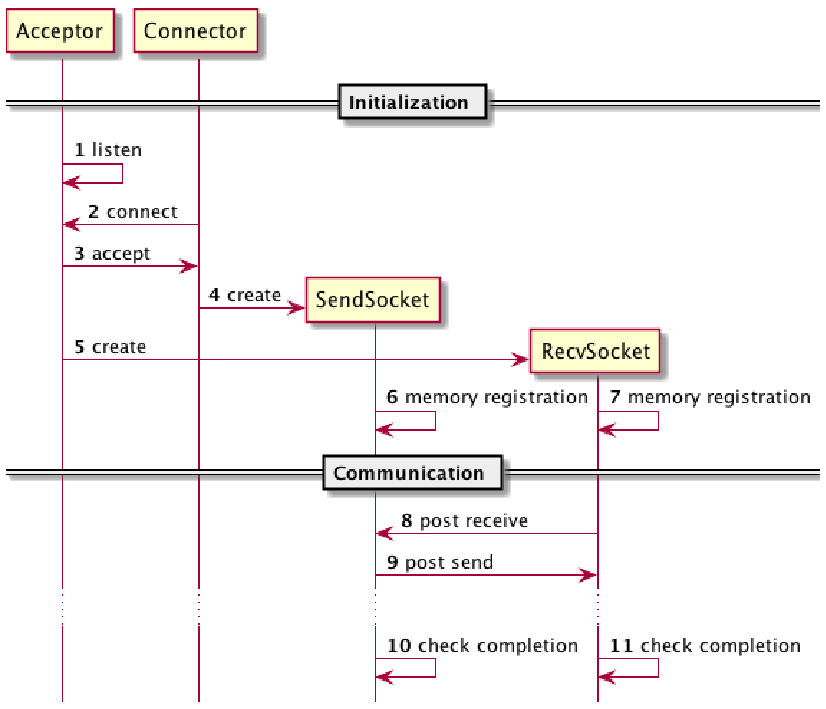


Fig. 3.8 Detailed description of the Communication Layer workflow

3. The Acceptor waits for an incoming connection.

4. Once the connection is established, the Connector returns a SendSocket object that allows to communicate with the connected endpoint through the created Queue Pair.

5. Once the connection is established, the Acceptor returns a RecvSocket object that allows to communicate with the connected endpoint through the created Queue Pair.

6. The SendSocket registers a memory area creating a Memory Region over it. This allows to perform RDMA operations on the registered memory.

7. The RecvSocket registers a memory area creating a Memory Region over it. This allows to perform RDMA operations on the registered memory.

8. The RecvSocket posts a memory buffer where data will be received. The memory buffer needs to be within the boundaries of a previously registered memory area and it cannot be modified until the receive operation is completed. Each operation of this type adds a Work Request to the Receive Queue of the RecvSocket's Queue Pair.

9. The SendSocket posts a memory buffer that will be remotely sent. This memory buffer needs to be within the boundaries of a previously registered memory area and it can not be modified until the send operation is completed. Each operation of this type adds a Work Request to the Send Queue of the SendSocket's Queue Pair.

10. The SendSocket polls the Send Completion Queue for completed send operations. Each operation of this type retrieves all the Work Completions present in the Send Completion Queue of the SendSocket.

11. The RecvSocket polls the Receive Completion Queue for completed receive operations. Each operation of this type retrieves all the Work Completions present in the Receive Completion Queue of the RecvSocket.

Steps 8 and 9 can be performed repeatedly without waiting for previous completions. When a send/receive operation is started by an HCA, a Work Request is automatically consumed from the Send/Receive Queue Pair and once an operation is completed a Work Completion is automatically added to the Send/Receive Completion Queue. The overlying software using the Communication Layer needs to take into account these mechanisms in order to find a balance, avoiding to empty or overfill Queue Pairs and Completion Queues.

3.4.2 TCP-based version

Performing RDMA operations needs specific RMDA-aware hardware which is not a commodity technology but it is available on HPC clusters only. This can complicate development and testing of applications that make use of verbs. In order to allow the execution of LSEB in any environment (virtual nodes, cluster with ethernet network, etc.) a TCP-based version of the Communication Layer has also been developed.

The API remains the same in both the versions of the Communication Layer, because all the RMDA concepts like Queue Pairs, Completion Queues, etc. are masked by the API itself. It is possible to choose which implementation to use at compile time between "TCP" and "VERBS" through the definition of a parameter named "TRANSPORT" passed to the cmake command.

The TCP-based version of the Communication Layer relies on the Boost.Asio library to perform network operations. Using an Acceptor-Connector like approach allows an easy adoption of the Boost.Asio Library, being itself based on that design pattern. Boost.Asio is a cross-platform C++ library mainly for networking and some other low-level input/output programming [42]. It has successfully abstracted the concepts of input and output that work not just for networking but for serial ports, files, and so on. On top of these, the developer can do input or output programming synchronously or asynchronously. The library is portable and works across most operating systems, scaling well over thousands of concurrent connections. The networking part was inspired by Berkeley Software Distribution (BSD) sockets; it provides an API that deals with Transmission Control Protocol (TCP) sockets, User Datagram Protocol (UDP) sockets and Internet Control Message Protocol (ICMP) sockets.

3.5 The Logic Layer

The Logic Layer is the upper layer of LSEB and it implements the design of the Event Builder discussed in section 3.2. It integrates the Readout Unit and the Builder Unit (described in sections 3.5.4 and 3.5.5 respectively) in a single process, running each unit as a separate thread. Referring to the Communication Layer design, the Readout Unit acts as the Connector and sends data through a SendSocket to the Builder Unit, that acts as the Acceptor and receives data through a RecvSocket. Besides the Readout Unit and the Builder Unit, the Logic Layer foresees other two components called Generator and Controller (section 3.5.3) for the generation and control of simulated fragments. These are required in order to allow the simulation of the PCIe40 readout boards during development and testing phases. A simplified schema of the separation between the two layers is shown in Fig. 3.9.

A special mention is deserved for the local communication, that is the the communication between the Readout Unit and its local Builder Unit. In LSEB both the Readout Unit and the Builder Unit belong to the same process: this means that they can communicate without relying on the network or even without an Inter Process Communication (IPC) mechanism. To

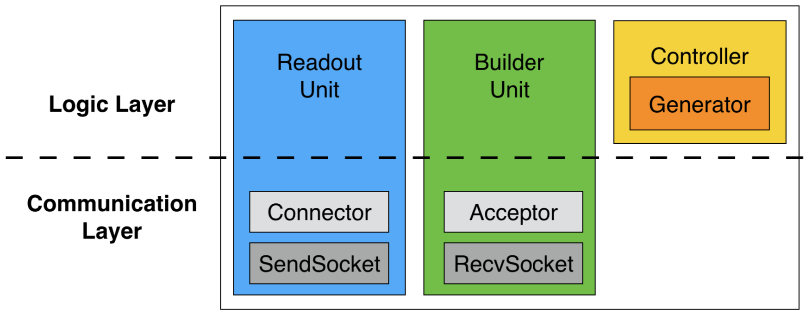


Fig. 3.9 Separation of Communication Layer and Logic Layer in LSEB

exploit the benefits of multithreading the Logic Layer implements the local communication in a different way, bypassing the Communication Layer and using two shared queues in order to exchange fragments between the Readout Unit and the Builder Unit.

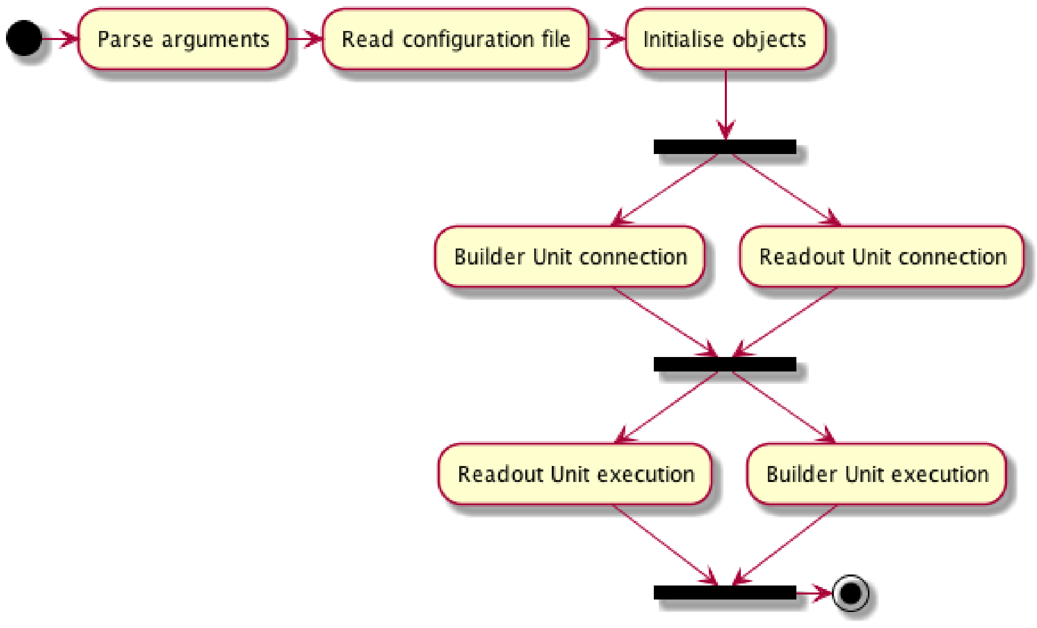


Fig. 3.10 Logic Layer workflow

The workflow of the Logic Layer, shown in Fig. 3.10, foresees the following steps:

1. The program parses the command line parameters. The only required parameter is the path of the configuration file. It is also possible to set parameters such as the directory where the log will be saved (the default behaviour is to log on standard output), the label that uniquely identify in the configuration file the node where the program is

running (the default one is the hostname) and the duration of the run in seconds (the default behaviour is an infinite run).

2. The program parses the configuration file. The configuration file specifies all the information needed to configure the Generator, the Controller, the Readout Unit and the Builder Unit. An example can be found in appendix B.1.
3. The Generator, the Controller, the Readout Unit and the Builder Unit are initialized.
4. The Readout Unit, running in a separated thread, starts to establish the connections with all the Builder Units.
5. The Builder Unit, running in a separated thread, starts to accept all the connections from the Readout Units.
6. The program waits until all the connections are established.
7. The Readout Unit, running in a separated thread, starts to acquire and send fragments. A more detailed description of this step can be found in section 3.5.4.
8. The Builder Unit, running in a separated thread, starts to receive fragments. A more detailed description of this step can be found in section 3.5.5.
9. The program waits the end of the run.

3.5.1 Scheduling strategy

In both Pull and Push protocols a supervisor may be present for deciding how to assign events to the Builder Units. Alternatively, a predefined scheduling strategy may be adopted by all the Readout Units or all the Builder Units (depending on the protocol). In the case of LSEB, that implements a Push protocol, a possible supervisor would need to continuously contact the Readout Units for communicating which Builder Unit has been elected for each event. This can cause on overhead in terms if software complexity, latency and network traffic. This overhead can be decreased assigning to each Builder Unit a set of events rather than a single one each time.

In the first implementation of LSEB it was decided to avoid the use of a central supervisor, preferring a predefined scheduling strategy shared by all the Readout Units. The adopted strategy is a round-robin pattern, in which the ith fragment is sent to the jth Builder Unit with $j = (i \mod N)$, where N is the number of nodes of the Event Builder. This choice reduces the points of failure and ensures more reactivity by the Readout Units, that do not need to wait a communication from the supervisor in order to know whom to send the fragments. Anyway, this does not preclude the adoption of a central supervisor in the future, in case a load balancing mechanism will be desired.

Fault tolerance

The fault tolerance is a property that enables a system to continue operating properly in the event of the failure of some of its components. Failures in the Readout Unit are a critical aspect and cannot be easily managed due to the physical connection of the PCIe40 cards. In this case the part of data coming from the detector to the failed node is lost until the failure is recovered. On the other hand, a failure of a Builder Unit can be managed more easily. A central supervisor can notice the failure of a Builder Unit, avoiding to communicate to the Readout Units to send events to that Builder Unit. However, the adoption of a central supervisor adds a single point of failure to the system that needs to be handled. The previously described round-robin pattern does not allow to dynamically change the policy of scheduling, in order to avoid sending events to unavailable Builder Units. Thus, in case of failure of a Builder Unit the Event Builder loses those events that should be sent to that Builder Unit. As an example in a 500-node Event Builder the failure of one node implies that:

- 1/500 of the all events is lost due to the unavailable Builder Unit
- for each event 1/500 of fragments is lost due to the unavailable Readout Unit

3.5.2 Data format

As soon as the Event Builder goes into production, each Readout Unit will read data written directly in RAM by the PCIe40 cards. At the time of writing the data format has not yet been defined and both the PCIe40 card and its driver are still in development [13]. Thus, in LSEB a plausible data format has been adopted. In this data format each event fragment is composed by:

- **ID.** The ID field contains the event identifier and it is common to all the fragments belonging to the same event.
- **LENGTH.** The LENGTH field represents the length of the fragment.
- **FLAGS.** The FLAGS field has a debugging purpose.
- **DATA.** The DATA field represents the real fragment that is the data coming from the detector and it has a variable length.

All the ID, LENGTH and FLAGS form the header of the fragment. In order to facilitate the identification of fragments performed by each Readout Unit a further data structure called metadata as been added. Each metadata element refers to a specific fragment and it is composed by:

- **ID.** The ID field is the same as the ID contained in the fragment header.
- **LENGTH.** The LENGTH field represent the length of the fragment.
- **OFFSET.** The OFFSET field is the distance in bytes from the beginning of the memory buffer to the beginning of the fragment.

Figure 3.11 shows the relationship between fragments and metadata: the metadata allows to easily identify the beginning and the end of a fragment or even a bulk of fragments without accessing the data in memory for reading each fragment's header.

Metadata size

The metadata size needs to be small enough to not affect the data transfer from the PCIe40 to the Readout Unit.

The event identifier should be unique within a single acquisition run. With an ID field of 48 bits at an acquisition rate of 40 MHz the event identifier will wrap after about 81 days,

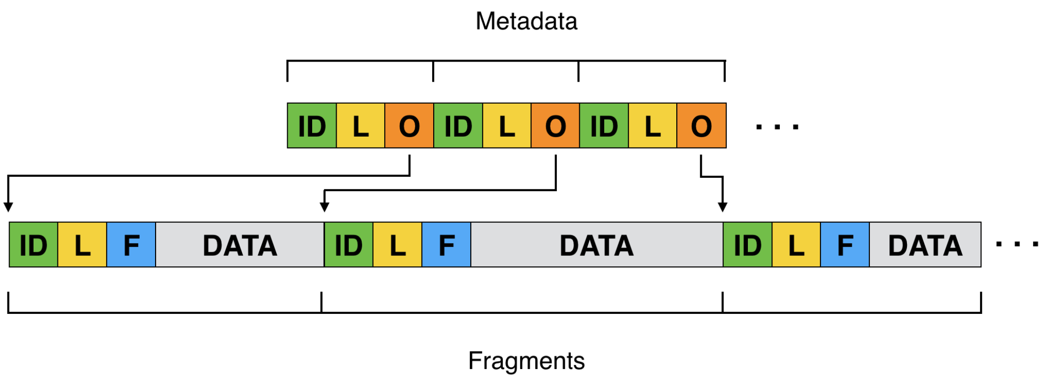

Metadata
ID L O ID L O ID L O · · ·
ID L F DATA ID L F DATA ID L F DATA · · ·
Fragments

Fig. 3.11 Relation between fragments and metadata

that is an acceptable maximum duration for a run. The size of the event identifier can be further reduced assuming that an identifier must be unique inside the event-building process instead of the whole run. In this case 32 bits can be enough, because the event identifier will wrap after about 107 seconds, and it is not expected that an event will remain inside the event-building process for so long. However, this solution requires an external mechanism that adds the remaining bits to the event identifier in order to allow to uniquely identify it within the run.

The data alignment of the fragments written by the PCIe40 is not yet defined. In reference [43] a data alignment of 32 bytes has been recently presented. Assuming this data alignment, the LENGTH field can safely represent the length of a fragment (about 200 bytes) using 4 bits. The OFFSET instead needs to be big enough to cover the whole memory buffer written by the PCIe40. In the current implementation of the PCIe40 this memory buffer is of 4 GB [35]. Thus, a possible size for the OFFSET field is 28 bits. The resulting metadata size is of 80 bits, or 10 bytes, that is about 5% of a single fragment. The metadata entry size can be further reduced incrementing the data alignment dimension, with the drawback of an increase of data fragmentation and wasted memory due to data padding.

3.5.3 Generator and Controller

In LSEB the data acquisition performed by the PCIe40 cards is currently simulated by a software component called Generator. The Generator handles two distinct memory ring buffers, one for the fragments and one for the metadata. In order to avoid delays caused by a slow runtime data generation, the Generator fills in advance the buffers with generated fragments and metadata and then reserves and releases them on demand.

The real fragments coming from the detector will not have fixed sizes due to different collisions and detected particles. The Generator simulates this behaviour assigning to each fragment a size based on a Gaussian distribution with mean and standard deviation read from a configuration file (appendix B.1). Also the maximum value of the fragment size can be configured through the configuration file.

A feature that can be useful for testing purpose is the tunable frequency of fragment generation. However, the Generator simply reserves and releases a requested number of fragments, if available. In order to implement this feature the Generator has been incapsulated inside an object called Controller that asks for fragments at the desired frequency. This frequency is read from the configuration file by the Controller. A second purpose of the Controller is to isolate the changes needed to switch from simulation to a production environment, once the readout boards will be ready. The design of LSEB foresees that only the Controller needs to be modified to retrieve data from the PCIe40 driver instead of from the Generator.

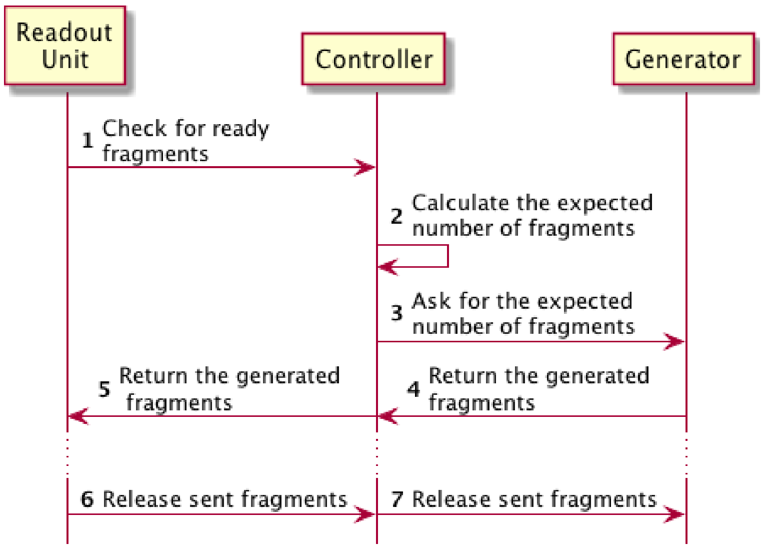

Readout Unit
Controller
Generator
1 Check for ready fragments
2 Calculate the expected number of fragments
3 Ask for the expected number of fragments
4 Return the generated fragments
5 Return the generated fragments
6 Release sent fragments
7 Release sent fragments

Fig. 3.12 Interactions between the Readout Unit, the Controller and the Generator

Figure 3.12 shows the interactions between the Readout Unit, the Controller and the Generator. Each time the Readout Unit asks for ready fragments to the Controller, this calculates the expected number of fragments. In order to do so, the Controller multiplies the total elapsed seconds from the beginning of the run with the expected frequency, obtaining the total number of expected available fragments. Then it compares the number of expected used fragments with the number of already generated fragments, asking to the Generator for the difference. Finally the Controller returns the new fragments, if any. Once the Readout Unit completes a send operation, it communicates to the Controller which fragments can be released and this information is propagated to the Generator that can reuse that memory for new available fragments.

3.5.4 The Readout Unit

The aim of the Readout Unit is to send ready fragments to the selected Builder Units at the maximum speed allowed by the network. The minimum buffer size needed to saturate the available bandwidth mostly depends on the network device. Different data rates and even HCAs with the same data rate but provided by different vendors may require different buffer sizes to saturate the bandwidth [44]. As an example, Figure 3.13 shows a benchmark performed on two nodes connected back to back with Mellanox FDR InfiniBand cards. In this configuration the minimum buffer size required is about 8 KB, far above the size of a single fragment. More detailed tests can be found in chapters 4 and 5.

Thus, considering that the size of a single fragment may not be enough in order to saturate the available bandwidth, the Readout Unit foresees to send fragments in bulks of configurable cardinality. This approach is similar to that of the adopted scheduling strategy (see section 3.5.1), that foresees to send a range of fragments with contiguous IDs to the same Builder Unit. The number of fragments that need to be sent to a specific Builder Unit and the number of fragments that are sent in a single send operation can be specified with a single parameter of the configuration file (appendix B.1).

The Readout Unit is aware of the memory areas containing metadata and fragments. However, the information exchange between the Readout Unit and the Controller is limited

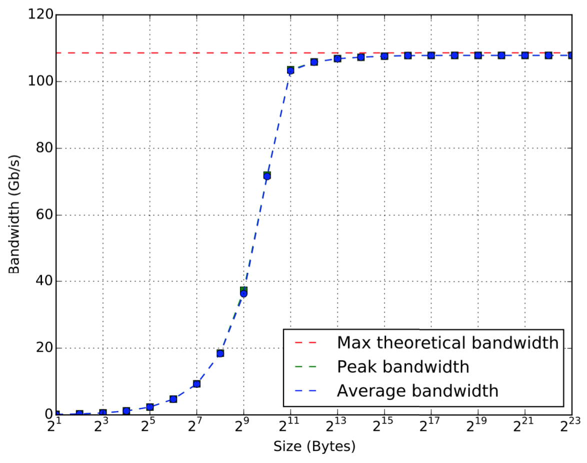


Fig. 3.13 Example of bandwidth banchmark

to the metadata, avoiding the access to the whole fragments in memory with related latency. This approach allows also a rapid identification of the beginning and the end of the memory area containing the fragments to be sent to a Builder Unit, without reading the header of each fragment. Despite that, the metadata are not sent to the Builder Unit. The Builder Unit needs to distinguish every single fragment for building a complete event. The advantage of reading consecutive metadata instead of not consecutive headers does not justify the overhead of sending remotely the metadata. As shown in section 3.5.2 this overhead can be estimated as at least about 5% of the throughput related to the fragments.

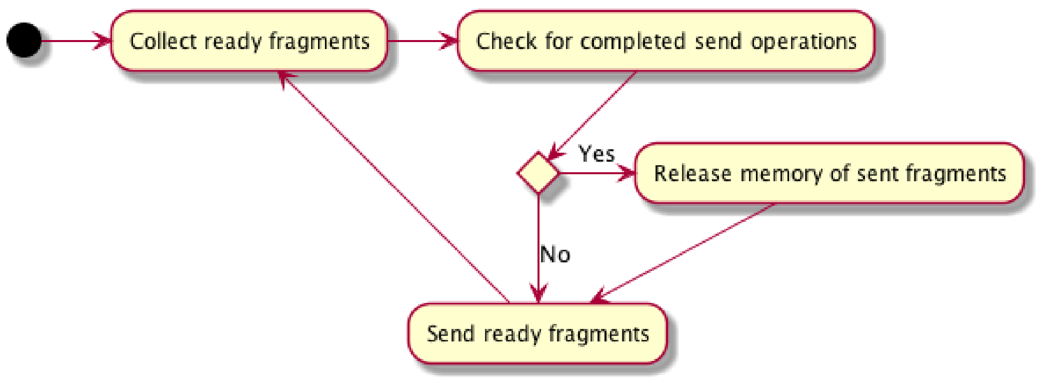

Collect ready fragments
Check for completed send operations
Yes
Release memory of sent fragments
No
Send ready fragments

Fig. 3.14 Readout Unit workflow

The workflow of the Readout Unit, shown in Fig. 3.14, foresees the following steps:

1. The Readout Unit asks the Controller for ready fragments.

2. The Readout Unit checks for completed send operations.
3. If there are send operations that are completed, the Readout Unit releases the memory related to those fragments. The Generator can then reuse the released memory for new fragments to be sent.
4. The Readout Unit sends all the ready fragments following the adopted dispatching policy, until there are available fragments and connections with available work requests.

3.5.5 The Builder Unit

The aim of the Builder Unit is to receive fragments by all the Readout Units and to forward them as completed events to the Event Filter Farm. In LSEB this last step is currently not considered. Instead of sending the completed events to the Event Filter Farm, the Builder Unit checks the correctness of the header of each fragment and then releases the events, reusing the memory for next incoming fragments.

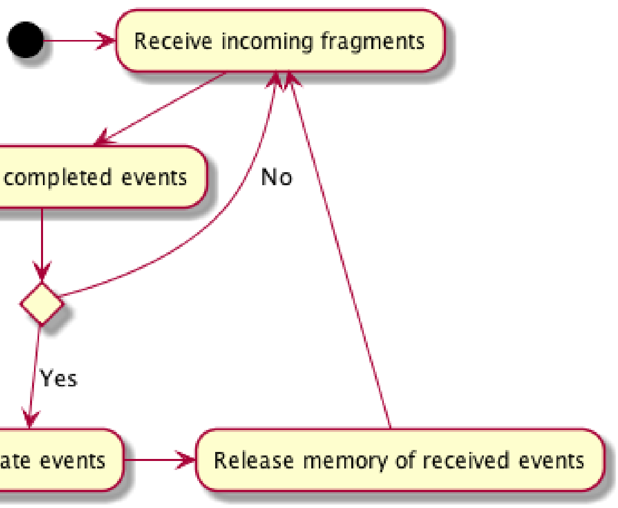

Receive incoming fragments
Check for completed events
No
Yes
Validate events
Release memory of received events

Fig. 3.15 Builder Unit workflow

The workflow of the Builder Unit, shown in Fig. 3.15, foresees the following steps:

1. The Builder Unit receives all the incoming fragments.
2. The Builder Unit checks if all the fragments of a given event have been received.

3. If there are complete events, the Builder Unit validates the header of each fragment, checking its fields.

4. The Builder Unit releases the memory related to the complete events. The Builder Unit can then reuse the released memory for the reception of new fragments.

3.6 CPU usage

As described in section 3.3.2, connections in LSEB are performed with the RDMA CM library that relies on blocking calls. A part from the connection phase, LSEB does not perform further blocking calls. Therefore, this means that monitoring the CPU with standard tools such as "top" results in a usage of 100% by each running unit. This does not mean that those cores is effectively used for useful operations, the threads are more likely spending most of the time waiting for data transfer completions. A profiling mechanism is implemented in LSEB to provide a rough evaluation of the effective core usage by both the Readout Unit and the Builder Unit. This mechanism measures the time spent by each unit performing useful operations (i.e. without waiting for some completions). Then the ratio between the measured time and the total elapsed time is logged together with other informations such as the average bandwidth.

Chapter 4

Performance and scalability tests

In this chapter the tests performed in different HPC clusters in order to study the performance and the scalability of the developed Event Builder software are described. Section 4.1 introduces those concepts that help to understand how performance can be typically improved on modern architectures. The methodology adopted for the tests is described in section 4.2 while the tests performed and related results are reported in sections 4.3 and 4.4. Finally, some considerations about the results obtained are given in section 4.5.

4.1 Relevant knowledge for system optimization

When running performance tests on modern servers several hardware features, such as the CPU frequency, the power management tools and the process affinity, must be taken into account in order to maximize the final attainable performance.

4.1.1 CPU Frequency Governor

One of the most effective ways to reduce power consumption and heat output on a system is to use the CPU frequency governor (CPUFreq). CPUfreq, also referred to as CPU speed scaling, allows the clock speed of the processor to be adjusted on the fly. This enables the system to run at a reduced clock speed to save power. The rules for shifting frequencies, whether to a faster or slower clock speed, and when to shift frequencies, are defined by the

CPUfreq governor. The governor defines the power characteristics of the system CPU, which in turn affects CPU performance. Each governor has its own unique behavior, purpose, and suitability in terms of workload.

The default is the "ondemand" governor, a dynamic governor that allows the CPU to achieve maximum clock frequency when system load is high, and also minimum clock frequency when the system is idle. While this allows the system to adjust power consumption according the system load, it does so at the expense of latency between frequency switching. As such, latency can offset any performance/power saving benefits offered by the "ondemand" governor if the system switches between idle and heavy workloads too often. For most systems, the "ondemand" governor can provide the best compromise between heat emission, power consumption, performance, and manageability. When the system is only busy at specific times of the day, the "ondemand" governor will automatically switch between maximum and minimum frequency depending on the load without any further intervention.

By contrast there is the "performance" governor, that forces the CPU to always use the highest possible clock frequency. This frequency will be statically set, and will not change. As such, this particular governor offers no power saving benefits. According to the "Mellanox performance tuning guide" [45] the performance of the InfiniBand cards can be improved setting the governor to "performance".

4.1.2 CPU idle states

The x86-architecture CPUs support various states in which parts of the CPU are deactivated or run at lower performance settings. These states, known as c-states, are numbered from C0 upwards, with higher numbers representing decreased CPU functionality and greater power saving. The c-states of a given number are broadly similar across processors, although the exact details of the specific feature sets of the state may vary between processor families. The more CPU units are stopped, reducing the voltage or even completely shutting down, the more energy is saved, but more is the time required for the CPU to "wake up" and be again 100% operational. Thus, enabling c-states generally results in a higher latency.

4.1.3 Non-Uniform Memory Access

Modern servers with two or more processors commonly have a Non-Uniform Memory Access (NUMA) architecture [46]. This means that there are different memory performance and latency characteristics when accessing memory directly attached to one processor, also called NUMA node, than when accessing memory directly attached to another processor in the same server (see Figure 4.1). Similarly, PCIe I/O devices are local to a specific processor and "remote" with respect to the others. To access a remote NUMA node the memory request must traverse the inter CPU link and use the memory controller associated to the remote NUMA node. This incurs a latency penalty on remote NUMA node memory access.

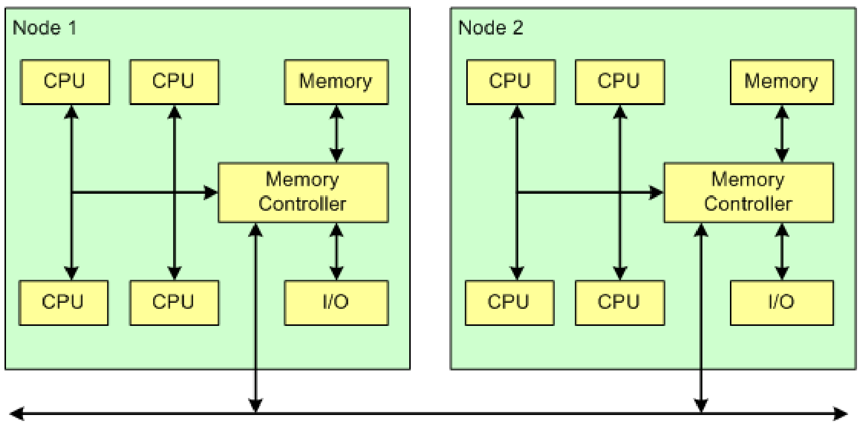


Fig. 4.1 Example of a Non-Uniform Memory Access architecture

In order to run an application on a certain NUMA node, the process affinity should be set using the "numactl" utility, which can control NUMA policy for processes or shared memory. For example, if the PCIe adapter's NUMA node is 1 and NUMA 1 cores are 8-15 then an application should run with process affinity that uses 8-15 cores only.

4.2 Tests methodology

Two different tests were performed for each cluster. At first, the available bandwidth between two nodes of the cluster is measured using a standard benchmark tool provided by the OFED package. After that, LSEB is run on a different number of nodes, depending on the size of the clusters.

4.2.1 OFED benchmark

Generically speaking, before testing an application on one or more nodes, it is required to execute standard benchmarks on those machines in order to establish how that application performs and how it can be improved. In case of applications that make use of RDMA, there is a set of micro benchmarks provided by the OFED package that allow to verify the effective point-to-point network capacity. One of these micro benchmarks, the so called ib_write_bw, was chosen and used to identify the real maximum bandwidth attainable between two random nodes belonging to the cluster. The tests performed with ib_write_bw foresee the execution of 5000 bidirectional RDMA write transactions for each different buffer size from 2^1 to 2^{22} bytes.

4.2.2 Tests with LSEB

The second test foresees to run LSEB on the cluster; depending on the number of nodes, several runs may be performed. The monitored parameters are the bidirectional bandwidth for each node and the core usage of the Readout Unit and the Builder Unit (see section 3.6). All the traffic related to the exchange of fragments between local Readout Units and Builder Units is not included in the bandwidth measured. In all the tests LSEB runs with the same fragment aggregation cardinality of 600 fragments, this implies that each data transfer has an average size of about 128 KB. This size was chosen after several benchmarks with different InfiniBand HCAs and is big enough to potentially saturate the bandwidth, as will be shown by the following benchmarks.

4.3 Cluster EDR InfiniBand at CERN

The LHCb experiment was able to test a first release of the ConnectX-4 adapter card [47], which supports the EDR 100 Gb/s InfiniBand. The testbed is composed of 4 nodes, each one connected through a ConnectX-4 adapter card to a EDR capable switch. Nodes are equipped with 2 x Intel Xeon E5-2650 v3 CPU running at 2.30 GHz, each of the CPUs have 10 cores and 20 threads and 64 GB of RAM. Each node is configured to get the best performance from the InfiniBand cards, such as setting the CPUfreq governor to "performance" and disabling c-states.

4.3.1 Standard benchmark

Benchmarks on this cluster have been made by the LHCb collaboration and presented during the 77th LHCb Collaboration Week [48]. A bidirectional bandwidth of 165.56 Gb/s has been obtained using the ib_write_bw tool. Given a theoretical bandwidth of 193.94 Gb/s foreseen by the EDR InfiniBand cards, the measured bandwidth is at the 85.37% level.

4.3.2 Results with the Event Builder software

On this cluster LSEB was tested performing the event-building with 2- and 4-nodes setups. The bandwidth as a function of time is plotted in Figure 4.2, showing a good stability over time both with 2- and 4-nodes. The average bandwidth is 156.85 Gb/s for the 2-nodes setup and 153.85 Gb/s for the 4-nodes setup, reaching 92.93% of the benchmarked bandwidth.

The measurements of the core usage performed by the internal profiler of LSEB demonstrate a load of about 23.87% for the Readout Unit and a load of 11.17% for the Builder Unit, as shown in Figure 4.3a and Figure 4.3b, respectively. Both these loads are stable over time and indicates that most of the core capacity is not yet used by LSEB.

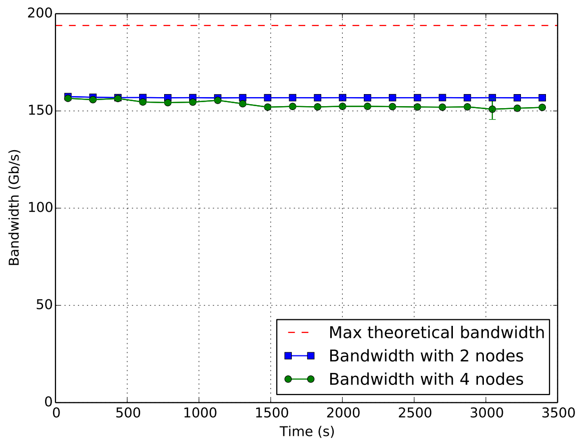
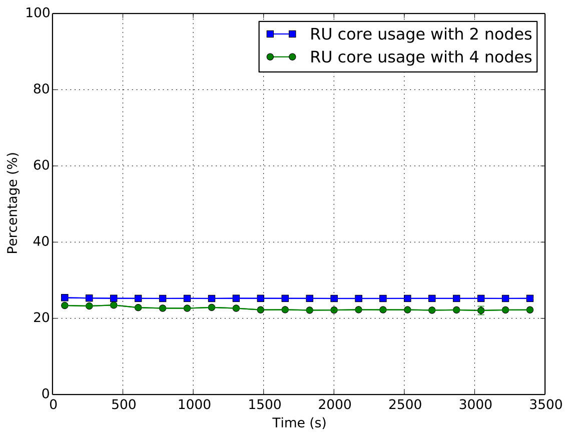
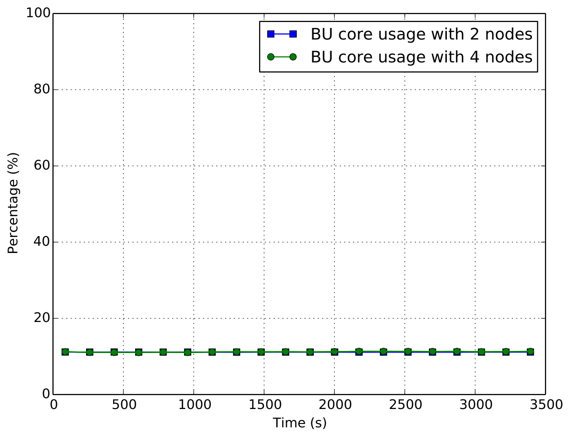


Fig. 4.2 Bandwidth running LSEB on nodes with EDR cards



(a) Readout Unit core usage



(b) Builder Unit core usage

Fig. 4.3 Builder Unit and Readout Unit core usage running LSEB on nodes with EDR cards

4.4 Cluster QDR InfiniBand at CINECA

Thanks to the contribution of the Italian National Institute for Nuclear Physics (INFN) [49], it was possible to perform scalability tests with LSEB on Galileo [50], one of the supercomputers hosted in CINECA [51]. CINECA is a non profit Consortium, made up of 70 Italian universities, 4 Italian Research Institutions and the Italian Ministry of Education. It offers support to the research activities of the scientific community through supercomputing and its applications. Galileo is the Tier-1 system of CINECA, introduced in January 2015. It is devoted to scientific computing on the basis of national and European proposals. This supercomputer is used to optimize and develop applications targeted at hybrid architectures, leveraging software applications in the fields of computational fluid dynamics, material and life science, and geophysics. The computing system is also available to European researchers as a Tier-1 system of the PRACE infrastructure [52].

Galileo is composed by 516 compute nodes, each one containing 2 Intel Xeon Haswell 8-core E5-2630 v3 processors, with a clock of 2.40 GHz. All the compute nodes have 128 GB of memory. 384 compute nodes are equipped with 2 accelerators (Intel Xeon Phi 7120P) for a total of 768 accelerators in the whole system. Each node is connected to a Infiniband network through a QLogic Single-Port QDR InfiniBand HCA [53]. Table 4.1 summarizes the system specific of Galileo.

Table 4.1 Galileo System Specific

Model	IBM NeXtScale
Architecture	Linux Infiniband Cluster
Nodes	516
Processors	2 x 8-cores Intel Haswell 2.40 GHz
Cores	16 cores/node, 8256 cores in total
Accelerators	2 Intel Phi 7120p per node on 344 nodes
RAM	128 GB/node, 8 GB/core
Internal Network	Infiniband with 4x QDR switches
HCA	QLogic Single-Port QDR InfiniBand

Due to the policies of the cluster it was not possible to perform all the fine-tuning operations described in section 4.1 in order to achieve best performances. Indeed for reasons related to power consumption, in Galileo each node has the governor set to "ondemand" and the c-states enabled. However, it was possible to take advantage of the process affinity using the "numactl" utility.

4.4.1 Standard benchmark

In Figure 4.4 the average bandwidth obtained running 10 times the ib_write_bw tool on two nodes of Galileo is plotted as a function of buffer size. It is important to note that the averaged bandwidth is affected by a significant statistical error due to the fact that different executions of ib_write_bw results in different measured bandwidths. Moreover, the average bandwidth and the peak bandwidth differ of about 15%. These behaviours are a symptom of a non-performant network, mostly caused by the constant presence of external jobs running on the cluster and a missing optimization setting on the nodes. Considering the average bandwidth, the benchmark reaches 42.88 Gb/s, that is the 78.82% of the maximum theoretical bandwidth expected with QLogic QDR InfiniBand cards (54.4 Gb/s).

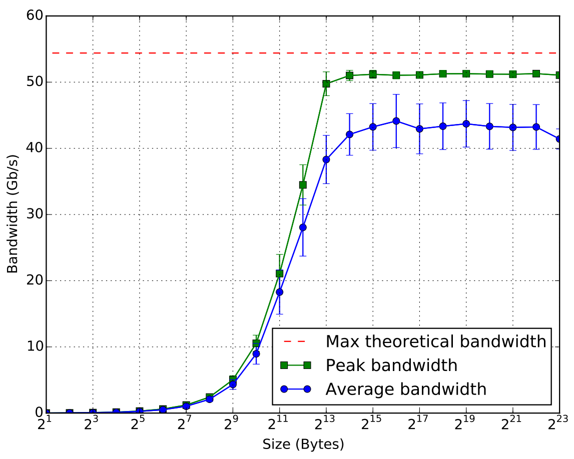


Fig. 4.4 Benchmark with ib_write_bw on Galileo: the blue line represents the resulting bidirectional average bandwidth and the green line represents the bidirectional peak bandwidth

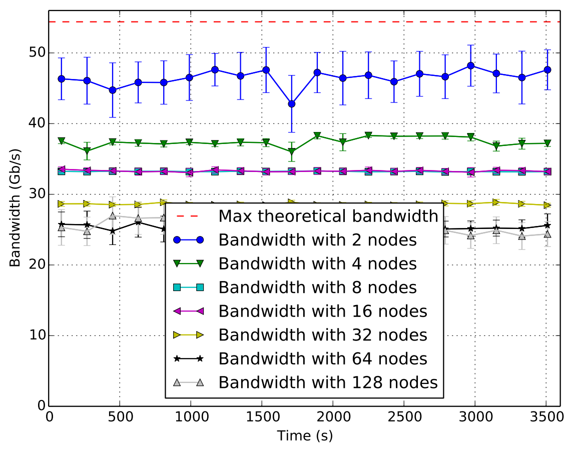
4.4.2 Results with the Event Builder software

As typical for clusters offering concurrent access to shared resources, Galileo computing nodes are accessible exclusively through a job scheduler. So, job execution is allowed by providing a list of computational requirements on a limited set of resources. For example, the requirements of a job can specify the number of needed cores to run on, with a maximum limit of 1024. Thus, although 500 nodes are available in Galileo, not all of them can be used concurrently by the same application. Even requiring 8 cores for each node instead of the 16 available, the maximum number of allocable nodes is 128. Reducing the number of required cores for each node may increase the number of allocable nodes but also increase the possibility to have external jobs running on the same nodes at the same time.

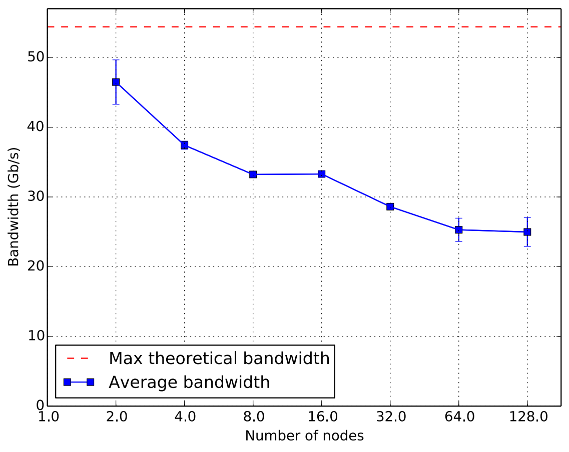
Starting from a 2-nodes setup, several tests were performed redoubling each time the number of nodes, up to the last test that involves 128 nodes. For the aims of the event-building a minimum of two cores is required: one for the execution of the Readout Unit thread and another one for the Builder Unit thread. However, in order to avoid the concurrent execution of external processes that may affect the performances of the acquisition, it is desiderable to reserve all the available cores on each node. For the 2-, 4-, 8-, 16- and 32-node tests the allocation of the entire node (16 cores each) was managed. Instead, for the 64- and 128-node tests only half of the available cores were allocated in each node.

The bandwidths measured running LSEB on Galileo in different nodes configuration are reported in Figure 4.5a as a function of time, presenting good stability over time. The scalability plot is shown in Figure 4.5b, where the average bandwidth is plotted as a function of the number of nodes: the bandwidth slowly decreases as the number of nodes increases, saturating at about the 58% of the benchmarked bandwidth 128 nodes.

The average core usage of the Readout Unit and the Builder Unit is plotted as a function of the number of nodes in Figures 4.6a and 4.6b, respectively. The core usage of the Readout Unit shows a decreasing trend as the number of nodes increases, which is caused by an increment of time spent to wait for the completion of transfer operations. On the other hand the core usage of the Builder Unit has an increasing trend, caused by an increment of the number of fragments that needs to be received and checked for each event. However, the Builder Unit does not saturate the core even with 128 nodes, reaching a load of about 30%.

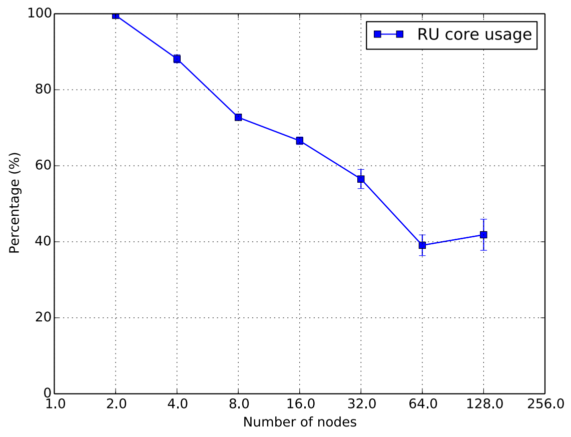


(a) Bandwidths measured

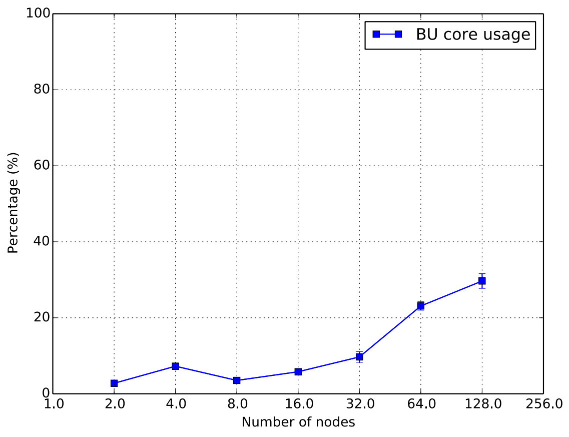


(b) Bandwidth scalability

Fig. 4.5 Bandwidths measured running LSEB on a different number of nodes on Galileo



(a) Readout Unit core usage



(b) Builder Unit core usage

Fig. 4.6 Builder Unit and Readout Unit core usage running LSEB on a different number of nodes on Galileo

4.5 Considerations

Two different clusters were used to test the Event Builder software. The first cluster is composed of 4 nodes connected together with EDR InfiniBand cards. Due to the low number of nodes, it is not the ideal candidate to test the scalability of LSEB. However, it allowed to check the performance of LSEB with the latest InfiniBand data rate standard with promising results. The second cluster is composed of about 500 nodes connected together with QDR InfiniBand cards. This kind of interconnects are not as performant as the EDR InfiniBand cards; moreover the cluster is tuned to save energy rather than to achieve best performance. Nevertheless, the presence of 500 nodes allow to perform scalability tests with a cardinality similar to that foreseen by the LHCb upgrade. Tests on this second cluster have shown that the LSEB scales up to 128 nodes the 60% of the available point-to-point bandwidth. Finally both the Readout Unit and the Builder Unit have most of the core capacity unused.

Chapter 5

Investigating low power architectures

With the increasing computer power consumptions and electricity costs, most data centers already spend about half as much for the electricity to power and cool their infrastructure as they do for the hardware itself, and this percentage can be expected to increase in the coming years. Thus, there are continuously R&D studies ongoing on upcoming low-power architectures. The work shown in this chapter is a very preliminary study in order to investigate a possible use of these upcoming technologies for the event-building purpose of LHCb and for high-throughput data acquisition in general. The section 5.1 describes the testbed foreseen for this study. Two distinct low-power x86 architectures are tested, comparing results to those obtained on a standard server. In sections 5.2 and 5.3 are shown the test performed and related results. Finally some considerations about the work done can be found in section 5.4.

5.1 Testbed setup

Measurements on low-power architectures are performed in collaboration with the Computing On SoC Architectures project (COSA) [54], that provided the testbed. The tested architectures are the Intel Xeon D-1540 [55] and the Intel Atom C2750 [56], two low-power x86 processors designed by Intel for its server line. For each architecture, the testbed is composed of two nodes of the same type connected back to back with InfiniBand HCAs. The available

InfiniBand HCAs were the QLogic QDR InfiniBand HCA [53] and the Mellanox FDR InfiniBand HCA [57]. The QLogic QDR InfiniBand HCA requires a PCI-2 slot with 8 lanes and its datasheet declares a maximum bidirectional bandwidth of 54.4 Gb/s instead of the 64 Gb/s foreseen by the standard (32 Gb/s for each direction). The Mellanox FDR InfiniBand HCA requires a PCIe-3 with 16 lanes and foresees a bidirectional bandwidth of 108.6 Gb/s. Depending on the PCIe capacity of each architecture, different InfiniBand HCAs were used.

The results are compared to those obtained on a testbed composed of two Intel Xeon based standard servers. These servers mount a dual socket Intel Xeon E5-2683 v3 [58], that was released by Intel in the 3rd quarter of 2014. This processor has 14 cores (with 2 threads per core) and it supports PCIe Gen 3 interconnects with a maximum of 40 lanes. The Thermal Design Power (TDP) of the Intel Xeon E5-2683 v3 is of 120 W. Due to the high PCIe capacity of this kind of processor, the Xeon servers support both the two available InfiniBand HCAs.

Tests performed in this study are similar to those described in section 4.2, with the addition of the power consumption monitoring while running LSEB. These measurements refer to the overall power consumption of each node including processor, motherboard, memory, disk and InfiniBand network card. Moreover, the benchmarks with ib_write_bw are performed using two different governor settings, the "ondemand" and the "performance" (for more details about governors see section 4.1.1). The comparison between the two different governors is needed to recognize if the change of the governor is affecting the performance of the system. The c-states feature (see section 4.1.2) was active in all the performed test because disabling it would result in a higher power consumption that is not in the aims of this study.

5.2 Intel Atom C2750

The Intel Atom C2750 was released by Intel in the 3rd quarter of 2013. It is a low-power System on Chip (SoC) designed for microservers and manufactured on 22nm technology. The Intel Atom C2750 has 8 cores without multi-threading and it supports PCIe-2 interconnects with a maximum of 16 lanes. The TDP of the Intel Atom C2750 is of 20 W. This processor is mounted on a Super Micro A1SAi-2750F motherboard [59] that provides a slot PCIe-2 with 8 lanes, an example of the motherboard is illustrated in Figure 5.1a. In this testbed the two Atom nodes are connected together with QLogic QDR InfiniBand cards (like the one shown in Figure 5.1b), due to the missing PCIe-3 support required by the more powerful Mellanox FDR InfiniBand cards.

(a) Super Micro A1SAi-2750F motherboard

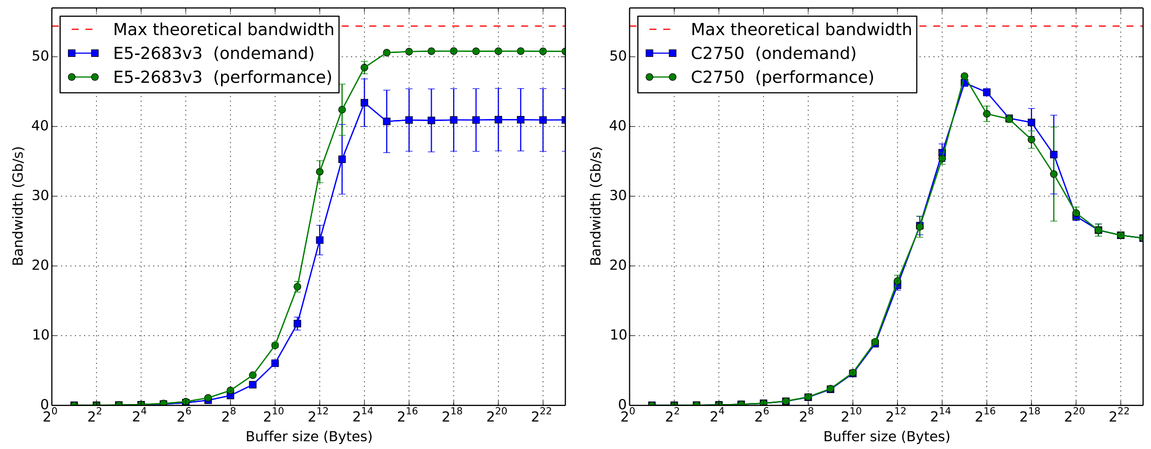
(b) QLogic InfiniBand QDR Host Channel Adapter

Fig. 5.1 Development board with Atom processor and InfiniBand QDR card

5.2.1 Standard benchmark

The results obtained running ib_write_bw on the Xeon servers with QDR InfiniBand connectivity are shown in Figure 5.2a, where the bandwidth is plotted as a function of the buffer size for both the governor settings. This benchmark confirms the behaviour observed testing LSEB on the Galileo cluster reported in section 4.4: setting the governor to "ondemand" does not allow to reach best performance and a stable bandwidth. On the other hand, a higher and more stable bandwidth is obtained setting the governor to "performance". In this configuration the maximum average bandwidth reached is 50.82 Gb/s, that is the 93.42% of the maximum theoretical bandwidth foreseen by this kind of interconnects.

On the Atom nodes the measured bandwidths present a relevant degradation with buffer sizes greater than 32 KB using both the governor settings, as shown in Figure 5.2b. This behavior can be caused by several factors such as CPU inefficiencies or network card issues with this platform. A further test with different network cards may help to fully understand this degradation in performance.



(a) ib_write_bw on E5-2683v3 (b) ib_write_bw on C2750

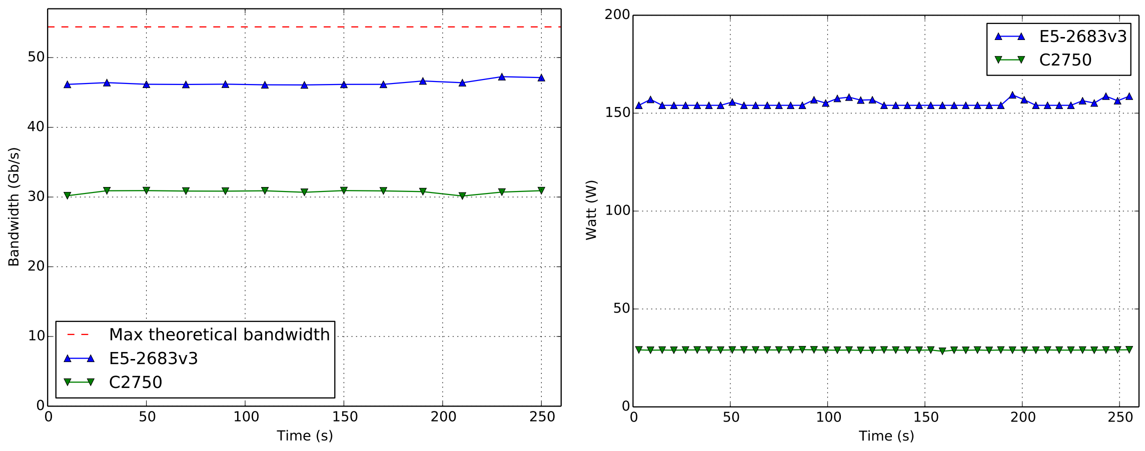
Fig. 5.2 Benchmark with ib_write_bw on nodes with QDR cards

5.2.2 Results with the Event Builder software

Apart from the benchmarks, the performance inefficiency of the Atom nodes is unveiled also running LSEB. Figure 5.3a shows that the Atom nodes reach an average bandwidth of 30.74 Gb/s with respect to the 46.38 Gb/s reached by the Xeon servers.

On the other hands the power consumption measurements turn the tables: the Atom nodes consume about the 18,73% with respect to the Xeon servers (28.93 W against 154.44 W), as illustrated in Figure 5.3b. A rough metric that could be used to compare these results is the Energy Consumption Rate (ECR) expressed in W/Gbps, that is the power consumption in Watt spent for each Gb/s of bandwidth. Following this metric the Atom nodes are three times more performant with respect to the Xeon servers, showing a ECR of 0.94 W/Gbps and 3.33 W/Gbps, respectively.

Table 5.1 summarizes all the measurements performed in this testbed. Besides the previously cited parameters, in the table are reported the power consumption in idle state and the maximum temperature reached in each processor type.



(a) Bandwidth (b) Power consumption

Fig. 5.3 Bandwidth and power consumption running LSEB on nodes with QDR cards

Table 5.1 Power consumption, temperature and bandwidth measured running LSEB on nodes with QDR cards

	E5-2683v3	**C2750**
Idle power consumption	77.46 W	18.20 W
EB power consumption	154.44 W	28.93 W
Max temperature	52.0 C	37.0 C
Average bandwidth	46.38 Gb/s	30.74 Gb/s

5.3 Intel Xeon D-1540

The Intel Xeon D-1540, also called XeonD in the following, brings the performance of Intel Xeon processors into a lower-power SoC. The Intel Xeon D-1540 was released by Intel in the 1st quarter of 2015. It is a low-power SoC designed for servers and manufactured on 14nm technology. The Intel Xeon D-1540 has 8 cores (with 2 threads per core) and it supports PCIe-3 (with max 24 lanes) and PCIe-2 (with max 8 lanes) interconnects. Being more a server than a pure low-power architecture, the Intel Xeon D-1540 has a higher power consumption with respect to the Intel Atom C2750, with a TDP of 45 W. This processor is mounted on a Super Micro X10SDV-F motherboard [60], that provides a PCIe-3 slot with 16 lanes, an example of the motherboard is illustrated in Figure 5.4a. In this testbed the two XeonD nodes are connected together with Mellanox FDR InfiniBand cards (like the one shown in Figure 5.4b).

(a) Super Micro X10SDV-F motherboard

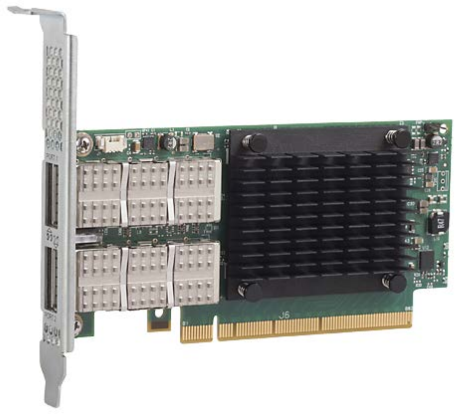
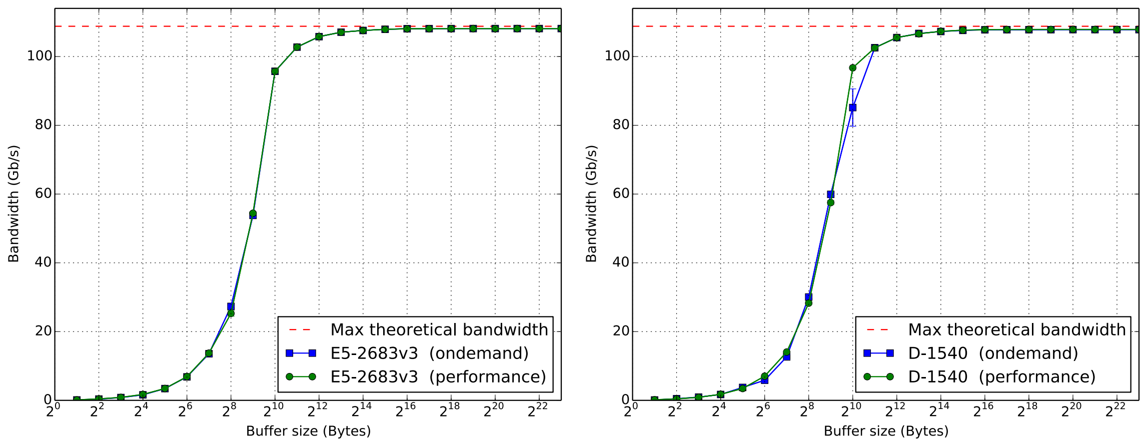

(b) Mellanox InfiniBand FDR Host Channel Adapter

Fig. 5.4 Development board with XeonD processor and InfiniBand FDR card

5.3.1 Standard benchmark

The benchmark performed shows that both the Xeon servers and the XeonD nodes can reach the maximum bandwidth available when using the Mellanox FDR cards, as illustrated in Figures 5.5a and 5.5b, respectively. The Xeon servers achieve the 99.57% of the theoretical bandwidth and the XeonD nodes reach the 99.35%. Furthermore, changing the governor settings does not affect the performance. This behaviour is mostly related to the InfiniBand cards, being the Xeon servers used in both the testbeds but with different interconnects.



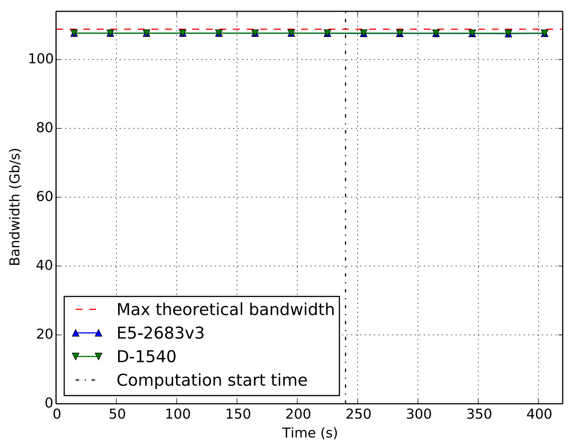
(a) ib_write_bw on E5-2683v3 (b) ib_write_bw on D-1540

Fig. 5.5 Benchmark with ib_write_bw on nodes with FDR cards

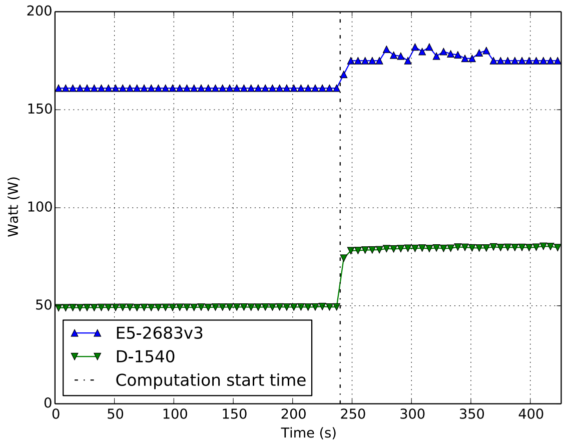
5.3.2 Results with the Event Builder software

The measurements of the bandwidth running LSEB software include the same tests performed with the Atom and an additional test which adds a pure computation process running over 4 cores on each node during the second half of the run. The aim of this added test is to investigate the possibility of the Event Builder to perform computations such as a pre-analysis of the events prior to send them to the Event Filter Farm.

The bandwidths as a function of time are shown in Figure 5.6a, where the black vertical line indicates the computation process start time. Both the Xeon servers and the XeonD nodes reach about the 99.12% of the theoretical bandwidth allowed by the FDR InfiniBand cards, keeping the bandwidth stable even with a computation process running. These performances were achieved with very different power consumptions, with the XeonD nodes that require about a third of the energy consumed by the Xeon servers, as illustrated in Figure 5.6b. Table 5.2 summarizes all the measurements performed in this testbed. Besides the previously cited parameters, in the table are reported the power consumption in idle state and the maximum temperature reached in each processor type.



(a) Bandwidth



(b) Power consumption

Fig. 5.6 Bandwidth and power consumption running LSEB on nodes with FDR cards

Table 5.2 Power consumption, temperature and bandwidth measured running LSEB on nodes with FDR cards

	E5-2683v3	**D-1540**
Idle power consumption	80.78 W	28.23 W
EB power consumption	161.00 W	49.02 W
EB power consumption with computation	176.54 W	79.12 W
Max temperature	56.0 C	59.0 C
Average bandwidth	107.63 Gb/s	107.65 Gb/s

5.4 Considerations

Each node of the Event Builder foreseen by the upgrade of the LHCb experiment requires at least two PCIe-3 slots of 16 lanes, one for the PCIe40 readout board and the other one for the network card. At the time of writing, none of the tested low-power architectures is suitable for the event-building of LHCb, due to the lack of enough PCI-e capability.

Despite that, the Intel Xeond D-1540 seems a really interesting processor for high-throughput data acquisition purposes. It ensures all the functionalities of the Intel Xeon family but reducing costs and power consumption. Tests show that with the adopted testbed this processor achieves comparable results in terms of bandwidth with respect to the standard server. Further studies on upcoming processors belonging to the Xeon D-1500 family should be performed in case Intel will upgrade the PCIe capacity of these products.

On the other hand, the Intel Atom C2750 can't compete with the high performance provided by standard Intel Xeon processors. Aside from the missing PCIe-3 support, tests show that this processor cannot achieve the maximum bandwidth allowed by the used HCA, even with the right PCIe slot. For these reasons this processor family cannot be a suitable candidate for the Event Builder. However, its extremely low power consumption makes it still interesting for data acquisition purposes where the bandwidth and computational requirements are less strict than those of the event-building of the LHCb experiment.

Conclusions

This thesis describes the study and implementation of an efficient Event Builder software for the LHCb upgrade based on the InfiniBand network technology. The developed software carries out the gathering of incoming data from a generator simulating the detector and performs the event composition in a completely trigger-less manner.

The Event Builder was successfully tested on two different clusters. The first cluster is composed of 4 nodes connected together with EDR InfiniBand cards. Due to the low number of nodes, it is not the ideal candidate to test the scalability of LSEB. However, it allowed to check the performance of LSEB with the latest InfiniBand data rate standard with promising results. Bandwidths measured on this cluster were stable over time, reaching 92.93% of the benchmarked bandwidth.

The second cluster is composed of about 500 nodes connected together with QDR InfiniBand cards. This kind of interconnects are not as performant as the EDR InfiniBand cards; moreover the cluster is tuned to save energy rather than to achieve best performance. Nevertheless, the presence of 500 nodes allow to perform scalability tests with a cardinality similar to that foreseen by the LHCb upgrade. Tests on this second cluster have shown that the LSEB scales up to 128 nodes reaching the 60% of the available point-to-point bandwidth. Finally both the Readout Unit and the Builder Unit have most of the core capacity unused.

A secondary activity related the study of the Event Builder is the investigation on x86 low-power architectures suitable for the event-building process foreseen by LHCb. Two different architectures are tested: the Intel Atom C2750 and the Intel Xeon D1540. The measured bandwidth and power consumption are compared with those obtained on standard servers. At the time of writing none of these architectures is a suitable candidate for high-throughput data acquisition applications; however, the results are still interesting especially when bandwidth and computational requirements are weaken and further investigations on upcoming x86 low-power architectures should be performed.

References

[1] "History of CERN." [Online]. Available: http://cern60.web.cern.ch/en/history

[2] L. Evans and P. Bryant, "LHC Machine," *Journal of Instrumentation (JINST)*, no. 3, 2008, ISBN 1748-0221.

[3] F. Bordry, S. Baird, K. Foraz, A. Perrot, R. Saban and J. Tock, "The first long shutdown (LS1) for the LHC," *Proceedings of 4th International Particle Accelerator Conference (IPAC)*, 2013, ISBN 978-3-95450-122-9.

[4] M. Bernardini and K. Foraz, "Long shutdown 2 @ LHC," *Proceedings of 4th International Particle Accelerator Conference (IPAC)*, 2015, DOI 10.5170/CERN-2015-002.

[5] A. Alves et al., "The LHCb Detector at the LHC," *Journal of Instrumentation (JINST)*, no. 3, 2008, ISBN 1748-0221.

[6] "LHCb web site." [Online]. Available: http://lhcb-public.web.cern.ch/lhcb-public/

[7] LHCb collaboration, *Letter of Intent for the LHCb Upgrade*. CERN, 2011, CERN-LHCC-2011-001. [Online]. Available: https://cds.cern.ch/record/1333091

[8] LHCb Collaboration, *Framework TDR for the LHCb Upgrade: Technical Design Report*. CERN, 2012, ISBN 978-9-29083-374-1.

[9] P. Moreira, "The GBT, a proposed architecture for multi-Gbps data transmission in high energy physics," *Proceedings of the Topical Workshop on Electronics for Particle Physics*, 2007, INSPIRE-774539.

[10] LHCb Collaboration, *LHCb Trigger and Online Upgrade Technical Design Report*. CERN, 2014, CERN-LHCC-2014-016.

[11] "PCI Express 3.0 specifications." [Online]. Available: http://www.pcisig.com/specifications/pciexpress/

[12] M. Bellato, G. Collazuol, I. D'Antone, P. Durante, D. Galli, B. Jost, I. Lax, G. Liu, U. Marconi, N. Neufeld, R. Schwemmer and V. Vagnoni, "A PCIe Gen3 based readout for the LHCb upgrade," *Journal of Physics: Conference Series*, 2014, DOI 10.1088/1742-6596/513/1/012023.

[13] P. Durante, N. Neufeld, R. Schwemmer, G. Balbi and U. Marconi, "100 Gbps PCI-Express readout for the LHCb upgrade," *19th IEEE-NPSS Real Time Conference (RT 2014)*, 2015, DOI 10.1088/1748-0221/10/04/C04018.

[14] “TOP500.” [Online]. Available: http://www.top500.org/

[15] “IEEE P802.3bs 400GbE Adopted Timeline,” September 2015. [Online]. Available: http://www.ieee802.org/3/bs/timeline_3bs_0915.pdf

[16] “InfiniBand Continues Growth and Interconnect Leadership on the TOP500,” November 2015. [Online]. Available: http://www.businesswire.com/news/home/20151116005422/en/EDR-100Gbs-FDR-56Gbs-InfiniBand-Continues-Growth

[17] R. Rosen, *Linux Kernel Networking: Implementation and Theory*. Apress, 2014, ISBN 978-1-43026-196-4.

[18] “Annex A 16: RoCE,” April 2010. [Online]. Available: https://cw.infinibandta.org/document/dl/7148

[19] “Annex A 17: RoCEv2,” September 2014. [Online]. Available: https://cw.infinibandta.org/document/dl/7781

[20] “A Remote Direct Memory Access Protocol Specification,” October 2007. [Online]. Available: http://www.rfc-base.org/txt/rfc-5040.txt

[21] “InfiniBand Architecture Specification - Volume 1,” March 2015, release 1.3. [Online]. Available: https://cw.infinibandta.org/document/dl/7859

[22] “InfiniBand Architecture Specification - Volume 2,” November 2012, release 1.3. [Online]. Available: https://cw.infinibandta.org/document/dl/7141

[23] H. J. R. Buyya, T. Cortes, *An Introduction to the InfiniBand Architecture*. Wiley-IEEE Press, 2002, ISBN 978-0-47054-483-9.

[24] “InfiniBand FAQ,” December 2014. [Online]. Available: http://www.mellanox.com/related-docs/whitepapers/InfiniBandFAQ_FQ_100.pdf

[25] “Compare of verbs implementation vs. the specifications.” [Online]. Available: http://www.rdmamojo.com/2013/11/23/compare-of-verbs-implementation-vs-the-specifications/

[26] M. Manzali, “lseb 2.0,” Mar. 2016. [Online]. Available: http://dx.doi.org/10.5281/zenodo.46935

[27] B. Stroustrup, *The C++ Programming Language*. Addison-Wesley Professional, 2013, ISBN 978-0-32156-384-2.

[28] “Boost C++ Libraries.” [Online]. Available: http://www.boost.org/

[29] “Is C a subset of C++?” [Online]. Available: http://web.archive.org/web/20080617183013/http://www.research.att.com/~bs/bs_faq.html#C-is-subset

[30] “CMake.” [Online]. Available: https://cmake.org/

[31] “Git.” [Online]. Available: https://git-scm.com/

[32] G. Liu and N. Neufeld, “DAQ Architecture for the LHCb Upgrade,” *Journal of Physics: Conference Series*, 2014, DOI 10.1088/1742-6596/513/1/012027.

[33] ——, “An event builder network for LHCb upgrade and the simulations on its performance,” *Journal of Physics: Conference Series*, 2011, DOI 10.1088/1742-6596/331/2/022016.

[34] ——, “The Generic Evaluation Tool for the LHCb Event Builder Network Upgrade,” *18th IEEE-NPSS Real Time Conference (RT 2012)*, 2012, DOI 10.1109/RTC.2012.6418195.

[35] P. Durante, N. Neufeld, R. Schwemmer, G. Balbi, and U. Marconi, “100 Gbps PCI-Express readout for the LHCb upgrade,” *Journal of Instrumentation (JINST)*, 2015, DOI 10.1088/1748-0221/10/04/C04018.

[36] A. Cohen, “A Performance Analysis of 4X InfiniBand Data Transfer Operations,” *Parallel and Distributed Processing Symposium*, 2003, ISBN 0-7695-1926-1.

[37] “The IPoIB driver.” [Online]. Available: https://www.kernel.org/doc/Documentation/infiniband/ipoib.txt

[38] P. MacArthur and R. Russell, “A Performance Study to Guide RDMA Programming Decisions,” *Proceedings of the 2012 IEEE 14th International Conference on High Performance Computing and Communication (HPCC)*, 2012, ISBN 978-0-7695-4749-7.

[39] F. Mietke, R. Rex, R. Baumgartl, T. Mehlan, T. Hoefler and W. Rehm, “Analysis of the Memory Registration Process in the Mellanox InfiniBand Software Stack,” *Proceedings of Euro-Par 2006 Parallel Processing*, 2006, ISBN 3-540-37783-2.

[40] “Tips and tricks to optimize your RDMA code.” [Online]. Available: http://www.rdmamojo.com/2013/06/08/tips-and-tricks-to-optimize-your-rdma-code/

[41] D. S. J. Coplien, *Pattern Languages of Program Design.* Addison-Wesley Educational Publishers, 1995, ISBN 978-0-20160-734-5.

[42] J. Torjo, *Boost.Asio C++ Network Programming.* Packt Publishing, 2013, ISBN 978-1-78216-326-8.

[43] “PCIe readout studies.” [Online]. Available: https://indico.cern.ch/event/291724/contribution/3/attachments/545336/751778/pcie40_update.pdf

[44] T. Schoenemeyer, H. El-Harake, “Performance Evaluation of Qlogic and Mellanox QDR InfiniBand,” 2011. [Online]. Available: https://www.researchgate.net/publication/268421638_Performance_Evaluation_of_Qlogic_and_Mellanox_QDR_InfiniBand

[45] “Performance Tuning Guide for Mellanox Network Adapters.” [Online]. Available: http://www.mellanox.com/related-docs/prod_software/Performance_Tuning_Guide_for_Mellanox_Network_Adapters.pdf

[46] D. Padua, *Encyclopedia of Parallel Computing.* Springer, 2011, ISBN 978-0-38709-766-4.

[47] "Mellanox ConnectX-4 adapter card." [Online]. Available: http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-4_VPI_Card.pdf

[48] "77th LHCb Collaboration Week." [Online]. Available: http://lhcbweek2015.bo.infn.it/

[49] "INFN." [Online]. Available: http://home.infn.it/en/

[50] "Galileo Cluster." [Online]. Available: http://www.hpc.cineca.it/hardware/galileo

[51] "CINECA." [Online]. Available: http://www.cineca.it/en

[52] "PRACE Project." [Online]. Available: www.prace-project.eu

[53] "QLogic QLE7340 datasheet." [Online]. Available: http://www.mullet.se/dokument/QLE7340_datasheet.pdf

[54] "INFN COSA Project - COMPUTING ON SOC ARCHITECTURE." [Online]. Available: http://www.cosa-project.it/home.html

[55] "Intel Xeon Processor D-1540." [Online]. Available: http://ark.intel.com/it/products/87039/Intel-Xeon-Processor-D-1540-12M-Cache-2_00-GHz

[56] "Intel Atom Processor C2750." [Online]. Available: http://ark.intel.com/it/products/77987/Intel-Atom-Processor-C2750-4M-Cache-2_40-GHz

[57] "Mellanox Connect-IB Single-Port InfiniBand HCA product brief." [Online]. Available: http://www.mellanox.com/related-docs/prod_adapter_cards/PB_Connect-IB.pdf

[58] "Super Micro X10SDV-F motherboard." [Online]. Available: http://ark.intel.com/it/products/81055/Intel-Xeon-Processor-E5-2683-v3-35M-Cache-2_00-GHz

[59] "Super Micro A1SAi-2750F motherboard." [Online]. Available: http://www.supermicro.com/products/motherboard/atom/X10/A1SAi-2750F.cfm

[60] "Super Micro X10SDV-F motherboard." [Online]. Available: http://www.supermicro.com/products/motherboard/Xeon/D/X10SDV-F.cfm

[61] "ISO/IEC 14882:2011." [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=50372

[62] "ISO/IEC 14882:2003." [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38110

[63] "ISO/IEC 14882:2014." [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=64029

[64] "ISO/IEC TR 19768:2007." [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=43289

[65] B. Schling, *The Boost C++ Libraries.* XML Press, 2011, ISBN 978-0-98221-919-5.

[66] B. H. K. Martin, *Mastering CMake.* Kitware, 2015, ISBN 978-1-93093-431-3.

[67] B. S. S. Chacon, *Pro Git.* Apress, 2014, ISBN 978-1-48420-077-3.

Appendix A

A.1 The C++ language and C++11 standard

C++ is a general-purpose programming language. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, including desktop applications, servers (e.g. e-commerce, web search or SQL servers), and performance-critical applications (e.g. telephone switches or space probes). C++ is a compiled language, with implementations of it available on many platforms and provided by various organizations.

C++11 [61] is one of the latest versions of the standard of the C++ programming language. It was approved by ISO on 12 August 2011, replacing C++03 [62], and superseded by C++14 [63] on 18 August 2014. The name follows the tradition of naming language versions by the year of the specification's publication, although it was formerly known as C++0x because it was expected to be published prior to 2010. Although one of the design goals was to prefer changes to the libraries over changes to the core language, C++11 does make several additions to the core language. Areas of the core language that were significantly improved include multithreading support, generic programming support, uniform initialization, and performance. Significant changes were also made to the C++ Standard Library, incorporating most of the C++ Technical Report 1 (TR1) libraries [64].

A.2 The Boost libraries

The Boost C++ libraries are a collection of modern libraries based on the C++ standard [65]. The source code is released under the Boost Software License, which allows anyone to use, modify, and distribute the libraries for free. The libraries are platform independent and support most popular compilers, as well as many that are less well known. The Boost community is responsible for developing and publishing the Boost libraries. The community consists of a relatively large group of C++ developers from around the world coordinated through the web site www.boost.org as well as several mailing lists. The mission statement of the community is to develop and collect high-quality libraries that complement the standard library. Libraries that prove of value and become important for the development of C++ applications stand a good chance of being included in the standard library at some point. The Boost community emerged around 1998, when the first version of the standard was released. It has grown continuously since then and now plays a big role in the standardization of C++. Even though there is no formal relationship between the Boost community and the standardization committee, some of the developers are active in both groups. The current version of the C++ standard includes libraries that have their roots in the Boost community. The Boost libraries are a good choice to increase productivity in C++ projects when the requirements go beyond what is available in the standard library. Because the Boost libraries evolve faster than the standard library, developers have earlier access to new developments, and they do not need to wait until those developments have been added to a new version of the standard library.

A.3 CMake

CMake is an extensible, open-source system that manages the build process in an operating system and in a compiler-independent manner [66]. Unlike many cross-platform systems, CMake is designed to be used in conjunction with the native build environment. Simple configuration files placed in each source directory (called CMakeLists.txt files) are used to generate standard build files (e.g., makefiles on Unix and projects/workspaces in Windows

MSVC) which are used in the usual way. CMake can generate a native build environment that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations. CMake supports in-place and out-of-place builds, and can therefore support multiple builds from a single source tree. CMake also supports static and dynamic library builds. Another nice feature of CMake is that it generates a cache file that is designed to be used with a graphical editor. For example, when CMake runs, it locates include files, libraries, and executables, and may encounter optional build directives. This information is gathered into the cache, which may be changed by the user prior to the generation of the native build files.

A.4 Git

Git is a widely used source code management system for software development [67]. It is a distributed revision control system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. Git was initially designed and developed in 2005 by Linux kernel developers (including Linus Torvalds) for Linux kernel development. As with most other distributed version control systems, and unlike most client–server systems, every Git working directory is a full-fledged repository with complete history and full version-tracking capabilities, independent of network access or a central server. Like the Linux kernel, Git is free software distributed under the terms of the GNU General Public License version 2.

Appendix B

B.1 Example of configuration file

The following is an example of a configuration file for the Large Scale Event Builder in the JSON format.

```
{
  "LOG_LEVEL": "NOTICE",
  "GENERATOR":
  {
    "MEAN": "200",
    "STD_DEV": "20",
    "FREQUENCY": "40000000"
  },
  "GENERAL":
  {
    "MAX_FRAGMENT_SIZE": "240",
    "BULKED_EVENTS": "600",
    "CREDITS": "20"
  },
  "ENDPOINTS":
  {
    "node001":
        {
                "HOST": "192.168.1.1",
                "PORT": "7000"
        },
        "node002":
        {
                "HOST": "192.168.1.2",
                "PORT": "7000"
        }
  }
}
```

Appendix C

C.1 The Communication Layer API

The following are the four objects that compose the Communication Layer with related API.

C.1.1 Acceptor

The Acceptor is the object used to listen for incoming connections and to accept them. Its member functions are:

- **Acceptor (int credits)**: The constructor requires the number of credits as parameter, that is the maximum number of pending (not already completed) operations allowed.
- **void listen (std::string hostname, std::string port)**: The method initiates a listen for incoming connection requests to the locally bound source address and port.
- **std::unique_ptr<T> accept ()**: The method retrieves the next connection request. If no requests are pending, it will block until a connection request is received. The returned value is a pointer to the newly created communication object (SendSocket or RecvSocket).

C.1.2 Connector

The Connector is the object used to ask for a new connection. Its member functions are:

- **Connector (int credits)**: The constructor requires the number of credits as parameter, that is the maximum number of pending (not already completed) operations allowed.
- **std::unique_ptr<T> connect (std::string hostname, std::string port)**: The method try to connect to the endpoint identified by address and port and return a pointer to the newly created communication object (SendSocket or RecvSocket).

C.1.3 SendSocket

The SendSocket represents a one-way communication channel and it is created by an Acceptor or a Connector when a connection is established. The SendSocket performs data transfer from the local endpoint to the remote connected one. Its member functions are:

- **void register_memory (void* buffer, size_t size)**: The method registers a memory buffer for RDMA operations.
- **std::vector<iovec> pop_completed ()**: The method return a vector of iovec that refer to buffers of completed send operations.
- **void post_send (iovec const& iov)**: The method starts a send operation and exits immediately.
- **int pending ()**: The method returns the number of send operations thar are currently not completed.

C.1.4 RecvSocket

The RecvSocket represents a one-way communication channel and it is created by an Acceptor or a Connector when a connection is established. The RecvSocket performs data transfer from the remote connected endpoint to the local one.Its member functions are:

- **void register_memory (void* buffer, size_t size)**: The method registers a memory buffer for RDMA operations.

- **std::vector<iovec> pop_completed** (): The method return a vector of iovec that refer to buffers of completed receive operations.
- **void post_recv (iovec iov)**: The method starts a receive operation and exits immediately.
- **void post_recv (std::vector<iovec> iov_vect)**: The method starts several receive operations (one for each iovec of the vector passed as parameter) and exits immediately.
- **std::string peer_hostname** (): The method returns the address of the connected peer.