Towards Virtualized 100+ Gbps Data Acquisition Software Systems

Jalal Mostafa, Suren Chilingaryan, Nicholas Tan Jerome, Andreas Kopmann, and Jürgen Becker

***Abstract*—Modern detectors in scientific infrastructures yield data at higher rates and in larger volumes. Despite the extensive data reduction on detector electronics, scientists employ software-based functions, e.g., high-level triggers, for further online data reduction that run on a dedicated computer cluster located at the scientific infrastructure's site. As a consequence, scaling, operating, and maintaining the stability and performance of this cluster becomes a demanding responsibility that requires extensive time and manpower. This paper proposes Data Acquisition Functions Virtualization (DFV), a new paradigm to minimize these operational efforts by eliminating computer clusters in scientific infrastructures and by running software-based online data reduction functions on widely available general-purpose campus computing facilities. DFV leverages computer virtualization and high-performance Ethernet networking to isolate software-based functions on campus computing facilities while sustaining their required input throughput. We explore the key technical challenges to realize DFV and propose the Data Acquisition Development Kit (DQDK), a novel framework for high-performance Ethernet-based readout functions and a cornerstone in DFV. The new data acquisition paradigm is applied to the TRISTAN upgrade of the KATRIN experiment at the Karlsruhe Institute of Technology. The framework can reduce CPU resources by a factor of 2.67x and save up to 15% of the consumed energy.**

***Index Terms*—AF_XDP, software-based data acquisition, high-throughput detectors, virtual machines, containers, computer networking**

I. Introduction

MODERN scientific detectors have high spatial and the temporal resolutions leading to an increase in data volumes and rates. For example, the raw data rates of the Compact Muon Solenoid (CMS) experiment at CERN have developed from $100\,\mathrm{GB/s}$ by end of 2013 to $200\,\mathrm{GB/s}$ by end of 2018 [1] and are expected to reach $50\,\mathrm{Tbit/s}$ with the High Luminosity Large Hadron Collider (HL-LHC) [2].

To handle the increasing data rates, scientists design hierarchical hybrid hardware-software data acquisition functions (DFs). Detector electronics, powered by technologies like Field Programmable Gate Arrays (FPGAs), are deployed adjacent to the detector and run trigger systems and event filtering algorithms to reduce the detector data rates. Because of their long test and development cycles, these technologies consume scientists' efforts and time to design and test data reduction DFs. Therefore, they are augmented with software-based DFs for further online data reduction on a computer cluster that is operated on the research infrastructure site. Contrarily, software-based DFs are easier to design and test complex reduction and feature extraction algorithms using high-level programming languages and parallel processing technologies like General Purpose Graphical Processing Units and the Message Passing Interface. Examples of software-based DFs are High-Level Triggers in CMS [3], ATLAS [4], and Belle II [5] or the online data reduction architecture in DUNE [6].

Since these computer clusters are located at the scientific infrastructure site, scientists have to put increasing efforts into managing and operating it. Such a cluster requires active system maintenance (e.g., system administration, software updates, etc.), which contributes to its availability and security. It is usually dedicated to one scientific infrastructure and is sporadically utilized when needed, e.g., for scientific measurements or simulations. For example, the Karlsruhe Tritium Neutrino (KATRIN) experiment, the Karlsruhe Research Accelerator, and the Far-infrared Linac and Test Experiment each have a dedicated computer cluster on the same campus at Karlsruhe Institute of Technology in Karlsruhe, Germany. This prevents sharing a computer cluster among several experiments and leads to resource under-utilization, plus redundant efforts and costs to commission and maintain each cluster. Another major concern is the cluster's scalability, which requires procuring new computer nodes to increase its computing power. Under these challenges, we say the traditional hybrid architecture suffers from "*The Operability Challenge*".

In this paper, we target overcoming the operability challenge by proposing **D**ata acquisition **F**unctions **V**irtualization (DFV), a new data acquisition (DAQ) paradigm that eliminates the need to operate a computer cluster in a research infrastructure. As shown in Fig. 1, the DFV paradigm exploits computer virtualization and Ethernet-based networking to run software-based DFs on campus computing facilities (CCF) that are abundantly available in most scientific and academic institutes. CCF are highly-scalable shared reusable computing resources managed and operated by the computing departments of the hosting institute. Using CCF offloads system maintenance to these departments and exploits their shared computing power. Computer virtualization and CCF's abundant computing resources provide easy means to scale DFs on demand without procuring additional hardware. As a result, multiple research infrastructures and scientific experiments can exploit CCF minimizing the efforts to operate software-based DFs.

Manuscript received DD MM 20YY; revised DD MM 20YY; accepted DD MM 20YY

Jalal Mostafa, Suren Chilingaryan, Nicholas Tan Jerome, and Andreas Kopmann are with Institute of Data Processing and Electronics, Karlsruhe Institute of Technology, 76344 Eggenstein-Leopoldshafen, Germany. (e-mail: jalal.mostafa@kit.edu).

Jürgen Becker, is with Institut für Technik der Informationsverarbeitung, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany. (e-mail: becker@kit.edu).

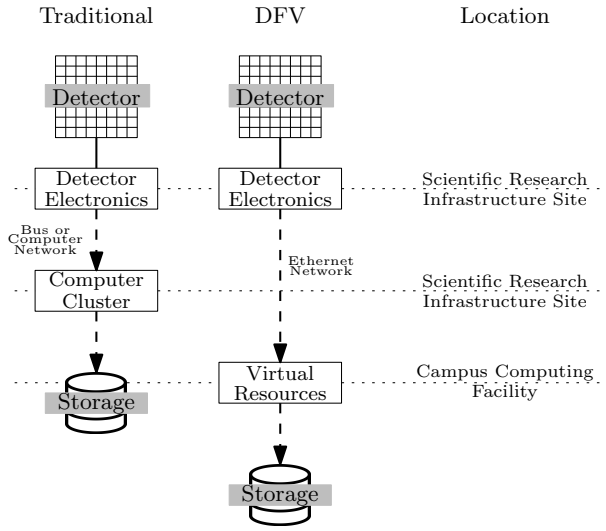

Fig. 1. DFV vs Traditional Hybrid Hardware-Software Data Acquisition

Since DFV should co-exist with diverse non-DAQ applications like mail and database servers on the CCF premises, it is crucial they do not compromise each other's execution. Diverse applications have diverse requirements each requiring an execution environment of specific software packages and services. To prevent breaking other applications's environments, DFV isolates applications through computer virtualization where every application is encapsulated in a virtual sandbox ensuring peaceful coexistence of applications. While computer virtualization isolates the execution environment, it does not prevent performance degradation caused by competition over computing resources among virtualized applications. On the contrary, computer virtualization despite its advantages degrades applications' performance due to the overhead added by the hypervisor [7], [8]. This is crucial for software-based DFs because degrading their performance could lead to potential loss of detector data while being received from hardware-based DFs.

To this end, this paper discusses the technical challenges of realizing DFV on CCF and proposes a design to overcome them. Our study is based on the flow of detector data from hardware-based DF by considering the requirements of modern detectors and the challenges imposed by: data transfer through computer networks, computing resources sharing on CCFs' computer systems, and finally computer virtualization overhead. We propose the Data acQuisition Development Kit (DQDK), an optimized software framework to design software-based DFs for DFV that exploits high-performance technologies to perform zero-loss Ethernet-based readout from detector electronics while minimizing the usage of shared computing resources. We quantify the impact of computer virtualization setups on DFV performance through DQDK. The TRItium Sterile Anti-Neutrino (TRISTAN) detector at the KArlsruhe TRItium Neutrino experiment is studied as use-case for DFV. Our results show that DFV can reduce CPU resources by a factor of 2.67x and save up to 15% of the consumed energy needed by the computer system.

II. Related Work

Computer virtualization in research infrastructures have attracted significant interest. However, it is only used for DFs that does not require high-performance like slow control systems or for offline data analysis services.

CERN operates its large-scale infrastructure, the large hadron collider (LHC), using the Worldwide LHC Computing Grid (WLCG), an international cooperative project incorporates 170 computing centers in 42 networks over a grid-based network infrastructure [9], [10]. The WLCG, however, operates distributed data management and analysis services of already saved data, but it does not operate the DAQ systems of the LHC where high bandwidth data from the LHC detectors need direct and online processing [10]. Instead, CERN operates online software-based DFs in a local computer cluster for every scientific experiment, e.g. CMS and ATLAS HLTs [3], [4]. In 2017, CERN outlined the computing infrastructure challenges to operate the DAQ systems of the LHC experiments in a virtualized computing platform [10]. Later in 2019, CERN published a roadmap to tackle these challenges and computer virtualization was a substantial part of it [11]. In the same year, the scientists of the ATLAS experiment at the LHC published a paper evaluating Kubernetes, a containers-based computer virtualization technology, as an orchestrator of the offline event filter computing resources of the DAQ systems [12]. The ALICE experiment at CERN has also started to adopt virtualized computing technologies in its infrastructure that are mainly for distributed data processing of already saved data [13].

The Karlsruhe Tritium Neutrino (KATRIN) experiment at Karlsruhe Institute of Technology (KIT) has operated a virtualized computing infrastructure called KaaS for its data management services and parts of the slow control system [14]. KaaS runs a container-based virtualization solution on only 3 computer servers, and it provides distributed data management services for KATRIN's slow control system. KaaS mainly focuses on data storage and offline data analysis and visualization.

Similar to KaaS, the the Jiangmen Underground Neutrino Observatory (JUNO) experiment in China is operating a service-oriented architecture for its slow control system using a Kubernetes-based computing platform that involves several distributed applications and middleware to guarantee the scalability and the availability of the slow control system of the experiment [15].

III. DFV Technical Challenges

This section discusses the technical challenges that undermine realizing DFV. Our discussion is based on analyzing the data flow of detector data from detector electronics to end of data reduction at software-based DFs.

The cornerstone of DFV is zero-loss data transfer of detector's data from detector electronics to virtual computing resources in CCF over Ethernet. Fig. 2 shows the flow of detector data in DFV. After being processed by frontend and backend detector electronics at the scientific infrastructure site, detector electronics have to prepare detector data for transfer to

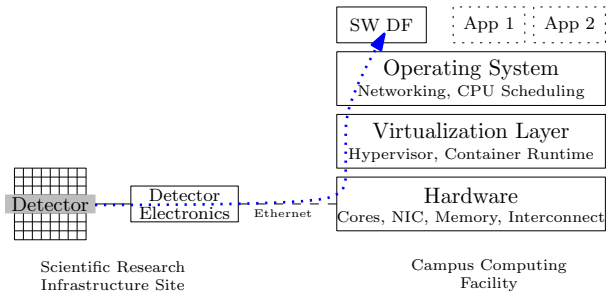

Fig. 2. Flow of Detector Data in DFV

the software-based DF over Ethernet in the CCF. At the CCF, pre-processed detector data is first handled by the physical general-purpose hardware represented the CPU cores, main memory, the network interface card (NIC), and the interconnects between them. The virtualization layer, e.g. virtual machine monitor (also known as hypervisor) or container runtimes, virtualizes the physical hardware resources into virtual resources to encapsulate every application into its sandbox so they do not interfere with each others' execution environments. Finally, the data goes through the operating system (OS) layer where implementations of computer networking software (networking sockets, Ethernet protocols, and NIC drivers) are implemented and sharing CPU cores between applications is managed through the CPU scheduler. Any bottleneck in these components can obstruct the data flow from detector electronics to the software-based DFs and lead to probable loss of detector data compromising the integrity of scientific data [16].

A. Involving Detector Electronics

DFV does not enforce major changes on detector electronics but only requires an implementation of Ethernet data transfer protocols for communication with the software-based DFs. However, it's only secondary to running their original tasks (trigger system, event building and filtering, etc.). Therefore, these implementations must be resource-efficient so they do not hamper developing their original DAQ tasks.

B. High-Throughput Resilient Networking

Traditional computer networking technologies do not support zero-loss $100+\,\mathrm{Gbit/s}$ data processing for modern scientific detectors [6], [17], [18]. A major reason for data loss is the *Slow Receiver* problem where the software-based DF (the data receiver) drops detector data because it cannot process the stream fast enough due to high data transfer latency [8], [19], [20]. Recent computer networking technologies, like RDMA and DPDK, can solve the *Slow Receiver* problem by offering lower data transfer latency. However, such technologies usually require special hardware that may not fit in CCF environments which are uncustomizable black boxes of hardware architectures, configurations, and software middlewares [21]. A good networking technology to establish DFV is one that does not only offer $100+\,\mathrm{Gbit/s}$ throughput but is also resilient to adapt to diverse technologies of CCF with no special hardware environment.

C. Shared General-Purpose Computing Resources

Unlike traditional hybrid hardware-software DAQ, CCF share resources among diverse applications. It is likely that these applications might interfere with each because they share the same physical CPU cores, hierarchical memory resources (main memory and CPU caches), and interconnects (memory bus and PCIe). This is especially true in CCF because are not tuned and optimized for high-performance applications like software-based DFs. For example, the CPU scheduling algorthim of the OS in CCF environments treat all applications alike without considering their workloads types and requirements. Such interference can trigger bottlenecks like the Memory Wall in general-purpose computer systems leading to degraded performance [19], [20], [22]. Tackling such complex technical challenges is hard and time-consuming intensifying the operational efforts of the scientific infrastructure.

D. Computer Virtualization

Computer virtualization is an integral component of DFV. It protects DFs from unwanted malicious accesses from other applications. However, isolation and security in virtual machines and containers have a performance cost on network-based applications like DFV due to overhead of virtualization features like Intel Virtual Machine Extensions [7], [23]. To overcome this overhead, different hardware and software network virtualization setups have been proposed for different applications like VirtIO-VHost (software network virtualization) and Single Root Input/Output Virtualization (hardware network virtualization). Understanding the performance of each virtualization setup is crucial to understand what is best to deploy DFV in CCF.

Considering these challenges to realize DFV, we argue that conventional data transfer technologies used in commodity computer networks are not reliable to realize DFV. An alternative reliable data transfer technology is required to realize DFV. Studying the challenges on shared general-purpose computing resources is required to propose the adequate tuning and optimizations to mitigate them. We need to study the performance impact of hardware and software network virtualization setups for DFV.

IV. DFV Design

To realize DFV and tackle the challenges raised in Section III, we set 3 design considerations inspired by the definition of DFV: *suitability for virtualization*, *resource efficiency on detector electronics*, and *high performance*.

DFV relies on computer virtualization available in CCF to run isolated DFs. However, computer virtualization constraints the virtual environment of its guests through abstraction layers on top of the computer host hardware. DFV can only have the virtualization merits if it respects the constraints of computer virtualization. Such constraints include: 1) *No Specific Hardware Requirements* for a specific high-performance networking technology as discussed in Section III-B, 2) *The Zero-Trust Security Model* of CCF refuses to provide applications with administrative or special privileges for security reasons, 3) *Compatibility with Other Applications* to prevent DFV from

obstructing the execution of other applications, e.g. depriving them of networking resources goes against the principles of computer virtualization.

DFV relies on detector electronics in the data transfer between the detector and the software-based DFs. This participation requires that data transfer protocols consume some resources on detectors' electronics. However, resources of detectors' electronics are considered to be scarce and expensive [18]. Hence, data transfer protocols implementation on detector electronics should be resource-efficient and leave a minor fingerprint on detector electronics resources.

To meet the requirements of modern detectors, DFV require processing $100+\,\mathrm{Gbit/s}$ data streams without data loss. Therefore, a networking technology that performs data transfer from detector electronics to software-based DF in DFV should meet the requirements of modern detectors.

Therefore, the 3 main pillars of DFV are: *Suitability for Virtualization*, *Resource Efficiency*, and *100+ Gbps Throughput*.

V. The Data acQuisition Development Kit

To realize DFV, we propose the Data acquisition Development Kit (DQDK) software framework. DQDK is a novel framework for high-performance Ethernet DAQ functions. It exploits high-performance networking technologies, parallel processing, and software optimizations that prevent contention due to resource sharing.

We build DQDK on top of AF_XDP, a novel low-latency data transfer technology. AF_XDP is a new type of networking sockets in Linux that enables the NIC drivers in the OS kernel to perform zero-copy data transfer directly to a user-space memory buffer (umem) allocated in the software-based DF. It promises 100+ Gbps data transfer and uses the resource-efficient TCP/IP protocols. AF_XDP is also considered suitable for virtualization because it is a 100% software solution native to the OS and its kernel allowing it to take advantage of tools that are already used to build clouds. Unlike other high-performance networking technologies that require special execution environments or resource-intensive protocols like user-space drivers [24] or RDMA [25], [26], it does not require a special hardware or execution environment and does not violate the zero-trust security model in campus computing facilities. AF_XDP also preserves the compatbility with other applications allowing traditional OS networking ones to coexist peacefully.

The DQDK framework is an easy-to-use lock-free multithreaded software framework optimized for scalability, cache awareness and compatibility. It employs the best practices to prevent performance degradation due to resource sharing in CCFs. The DQDK framework is implemented in $\sim$3200 lines of code, and is written in C and eBPF and utilizes the `libbpf` and `libxdp` libraries to run and manage AF_XDP sockets, the `libnuma` library for NUMA-aware processing, and the `libpthread` library for lock-free multithreaded processing.

A. Architecture

Fig. 3 shows the architecture of the DQDK framework. The DQDK framework employs threads, hereby called workers, where each worker is running a AF_XDP socket. A worker receives data through the AF_XDP socket and then executes a user-supplied DAQ logic over the received data. The workers are initialized and operated over 2 phases or paths: *The Data Path* and *The Control Path*. The data path is responsible for high-performance delivery of detector data from the network link to the DF, and the control path is responsible for configuring and setting up the worker and binding it to the data path. The control path is managed by an application programming interface (API) that initializes and passes the execution parameters to the internal components, e.g. number of workers. It has 2 main responsibilities: allocation of computing resources necessary for the data path, and loading the data path components to the system to start the DAQ process. The memory and CPU resources are allocated through the NUMA-Aware Memory Allocator and the CPU Manager respectively. The NUMA-Aware Memory Allocator allocates the umem for each worker in the data path from the local *huge pages* of the automatically detected NIC NUMA node. The memory allocator has 3 advantages: 1) NUMA-Aware Allocations: it allocates local memory and reduces cache misses resulting from allocating memory on remote non-local NUMA nodes; 2) Huge Pages-based allocations: it prevents high latencies resulting from frequent translations of virtual addresses into physical memory addresses; 3) Locked Memory: it prevents the operating system from swapping memory from main memory to far storage disks and thus causing performance degradation by violating the Principle of Locality. The CPU Manager is responsible for allocating and pinning CPU resources for the DQDK workers. It auto-detects the hardware configuration of the owner NUMA node and its CPU cores, it allocates for each worker 2 CPU cores (one for the application and another for the corresponding NIC interrupt handler). The performance advantages of the CPU manager are: 1) NUMA-aware CPU cores allocation that prevent cache misses due to remote memory accesses by allocating CPU and memory on the owner NUMA node; 2) it prevents interference from other tasks or the OS processes by isolating and dedicating the CPU core to the DQDK framework; 3) it protects memory locality by preventing frequent process context switching by configuring the OS's process scheduler to pin the DQDK framework threads only to the chosen dedicated CPU cores. After allocating resources, the DQDK API starts to load the data plane by creating the workers, the AF_XDP sockets and other components.

The data plane in DQDK consists of 4 main components: Workers, SIMD-powered UDP/IP, Batch Processor, AF_XDP Sockets, and the Forwarder. The forwarder preserves compatibility with other software applications by filtering detector data from other network data, e.g. data that belongs to other applications like slow control software in scientific infrastructure. If received network data is detector data, it is directly forwarded to the AF_XDP sockets in the user-space as seen in Fig. 3. Otherwise, the forwarder returns the ownership of this network data to the OS which ensures it is received by its target application. The sockets are polled infinitely for new data using the Batch Processor which takes care of managing the AF_XDP descriptors and passing data for protocol processing.

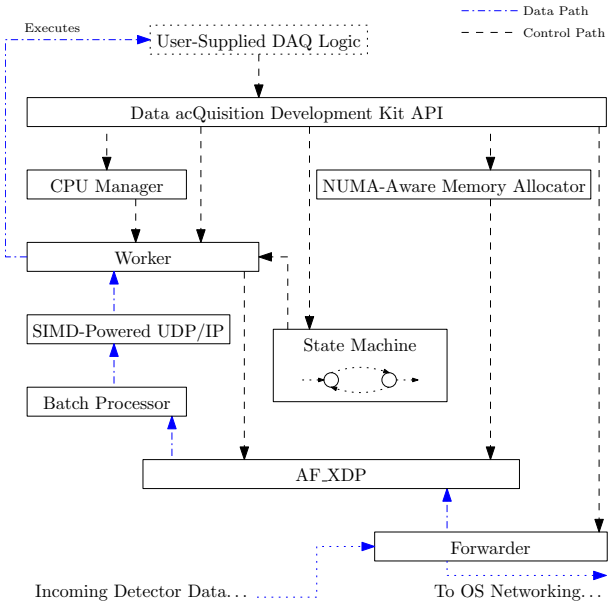

Fig. 3. The DQDK Framework Architecture

The DQDK data plane uses the User Datagram Protocol (UDP) and the Internet Protocol version 4 (IPv4) protocols for data transfer. The choice of the UDP/IP protocol stack is based on the protocols' lightweight implementations on detector electronics [27]–[29]. There exists network protocols that are lossless protocols like the Transmission Control Protocol (TCP) or Infiniband, but they are resource-demanding when implemented on detector electronics. Moreover, this does not mean UDP does not provide any mechanisms to detect data corruption in a distributed infrastructure like DFV.

The transition from the control path to the data path is maintained through a state machine. It ensures the data path is ready for processing detector data before the detector electronics starts sending them. This is achieved through a minimal protocol that synchronizes the slow control system and DQDK. The protocol should be implemented in the slow control system that is responsible for commanding detector electronics to start the scientific experiment and sending data.

B. Limitations

The design of DQDK framework has some limitations. For example, DQDK requires the Ethernet protocol and does not support a diverse set of data transfer protocols (e.g. TCP, IPv6) other than UDP and IPv4, but it is open to extension. The current IPv4 implementation also does not support packet fragmentation which requires manually increasing the Ethernet MTU when receiving data that is larger than the usual MTU (i.e. $1514\,\mathrm{B}$). It does not support sending Ethernet frames larger than $4\,\mathrm{kB}$. The DQDK framework uses CPU cores (2 cores per AF_XDP socket) to achieve zero-loss DAQ, and consequently requiring higher throughput may require scaling DQDK, increasing the number of the needed CPU cores.

VI. The TRISTAN Use-case

To prove its applicability, we apply DFV on the Tritium Sterile Anti-Neutrino (TRISTAN) detector upgrade from the KATRIN experiment [29]. It wants to explore the hypothetical existence of the so-called sterile neutrinos experimentally which are considered to have a potential contribution to dark matter. The TRISTAN detector is formed of independent modules called tiles. Each tile has 166 Silicon Drift Detector (SDD) hexagonal cells. The detector will be deployed at the KATRIN infrastructure in 2 phases: the first phase employs 9 tiles, the second employs 21 tiles.

The tiles will collaboratively work in 4 modes: Waveform, List-wave, List-mode, and Histogram. The Waveform Mode is a high throughput mode where the detector issues a raw and unfiltered data stream of the detector signal for a certain duration of all available tiles. The waveform will be sampled at $62.5\,\mathrm{MHz}$ with a sample size of $16\,\mathrm{bit}$ and producing $125\,\mathrm{Mbit/s}$ per channel. The rate of the detector in this mode is expected to reach $200\,\mathrm{Gbit/s}$ which should be collected *without loss* for at least $100\,\mathrm{ms}$ and requires at least $2.5\,\mathrm{GB}$ of memory. Longer runs can reach up to $1\,\mathrm{TB}$ of memory. The List-Wave is similar to the Waveform Mode but it includes triggered raw data. Similarly, the Histogram and the List-Mode modes are triggered modes but do not include raw data. In the Histogram Mode, a single detector tile is expected to trigger 16.6 million energy events per second to be collected, processed, and summarized in 4 to 10 histograms per detector channel before being saved to disk. Each histogram has 32768 bins of $4\,\mathrm{B}$ integers. Calculating the total required memory to build the histogram in the memory would result in $1.7\,\mathrm{GB}$ at minimum and $4.25\,\mathrm{GB}$ at maximum.

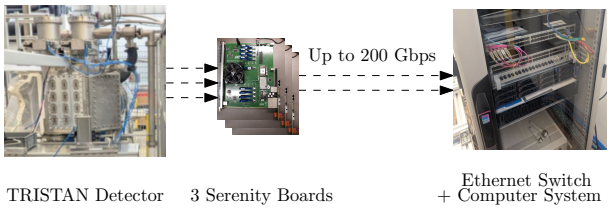

Fig. 4. Proposed Hybrid TRISTAN DAQ

Fig. 4 shows the proposed setup of the TRISTAN DAQ system for 9 tiles in phase 1. The 9 tiles are managed by 3 Serenity boards that acquire data from the detector channels and send them to a computer system where software-based DAQ is performed. Each Serenity board is connected to 3 tiles and is responsible for the event building and filtering of 498 channels. Detector data from all Serenity boards is aggregated in 2x $100\,\mathrm{Gbit/s}$ uplink *Ethernet* links to the computer system using an Ethernet switch.

VII. Experimental Setup

We test our software-based DAQ function for the TRISTAN detector by employing a simulation setup consisting of a Serenity-A2577 board equipped with a Xilinx VU9P FPGA & an enterprise-grade server. Fig. 5 shows the simulation setup. A Serenity-A2577 board implements a UDP client core including minimal IPv4 & ARP implementations over MAC & PMA/PCS Xilinx IP core. The UDP implementation is resource-efficient as it uses 0 block RAMs and consumes 1048 of the available 67200 Configurable Logic Blocks (roughly

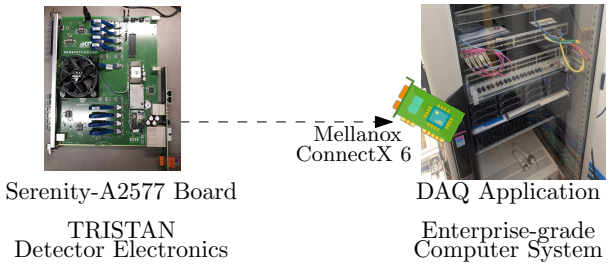

Fig. 5. TRISTAN DAQ Simulation Setup

1.5%). The UDP server on the board generates UDP traffic with 3392 B of packet size simulating the TRISTAN detector. The UDP data is then transmitted over a 100 Gbit/s link to the enterprise-grade server. It has 1 TB main memory balanced over 4 NUMA nodes. Each NUMA node has a 16-core Intel(R) Xeon(R) Platinum 8444H CPU @ 2.9GHz with Level 3 (L3) cache of 180 MB.

The system has a 100 Gbit/s NVIDIA-Mellanox ConnectX-6 Dx NIC connected over PCIe 4.0 x16 to the PCIe Root Complex of NUMA node 0. It runs TRISTAN's software-based DF for the Waveform and the Histogram modes. We provide 2 implementations for the software-based DF: one that is based on DQDK and another that is based on state-of-the-art traditional OS networking (UDP Sockets). Since we plan to deploy the software-based DAQ function on a container-based platform in the future, we perform our evaluation using containers with host networking in comparison to bare metal (no virtualization).

VIII. Performance Evaluation

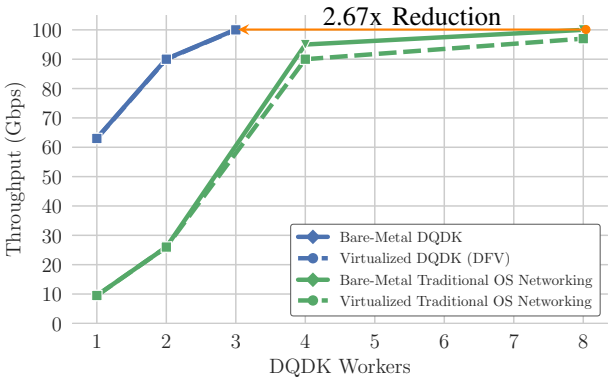

Fig. 6. Waveform Mode Throughput (Gbps) as function of DQDK Workers

Fig. 6 shows the throughput in Gbps without loss using DQDK as function of DQDK workers. The plot studies the required DQDK workers, and consequently the CPU cores, to process all the 100 Gbit/s traffic of the Waveform Mode with zero-loss. Using 1 worker, the maximum possible throughput with zero loss in DQDK is only 63 Gbit/s while it is only 9.5 Gbit/s with traditional OS networking. Increasing the throughput beyond 63 Gbit/s would push the NIC to drop detector data as there are not enough computing resources to process all received packets. The throughput of the DQDK implementation increases to 90 Gbit/s using 2 DQDK workers (4 CPU cores) and only 26 Gbit/s with traditional OS networking. It is enough to employ 3 DQDK workers (6 CPU cores) to saturate the link using the DQDK implementation, but we need up to 8 DQDK workers (16 CPU cores) to do the same if we use traditional OS networking. We notice that virtualization does not impact our DQDK implementation (2 coinciding blue plots in Fig. 6). On the other hand, we see that using virtualization for traditional OS networking impact their performance: even using the maximum number of DQDK workers (16 CPU cores) cannot process all the NIC bandwidth achieving only 97 Gbit/s.

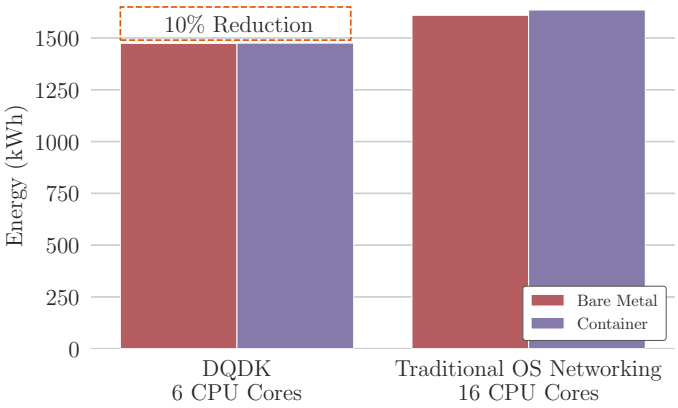

Fig. 7. Consumed Energy for Zero-loss 100 Gbit/s Setups

We also compare the consumed energy of both setups that can process all the 100 Gbit/s link: DQDK implementation with 3 DQDK workers and traditional OS networking implementation with 8 workers. Our DQDK implementation consumes 1474 kWh and the traditional OS networking implementation consumes 1609 kWh. Using containers will consume slightly more energy in the traditional OS networking implementation (~26 kWh). The energy savings are up to 161 kWh (around 10% less energy).

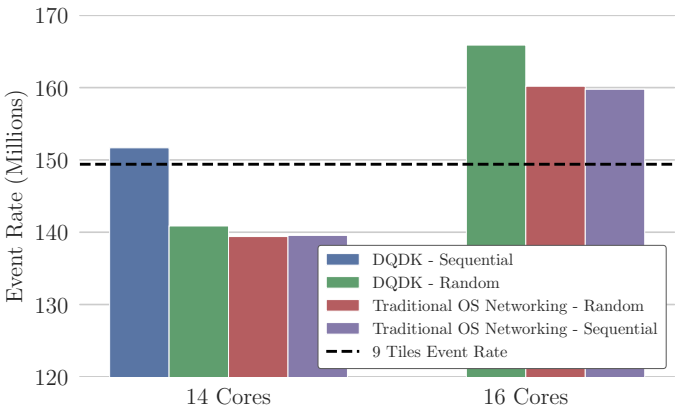

Fig. 8. DQDK Histogram Event Processing Rates

To simulate the Histogram mode on the Serenity board, we implement a simulator for TRISTAN Tiles (166 channels generating 16.6 million events per second for each tile) where each channel generates random numbers representing energy values. The goal is to calculate what are the needed resources

to process the energy events of 9 Tiles in Phase 1 of the TRISTAN deployment. We run the software-based DF inside a container. Similarly to the Waveform Mode, the Histogram Mode is using 3392 B of 212 16 B-events each. The required processing rate for all 9 tiles would be 149.4 million events per second.

The histogram in the software-based DF is allocated as a multidimensional array. Incrementing the histogram bins based on the received events requires memory accesses to this array and yielding cache misses in some cases. Cache misses will increase the dispatching latency and the processing latency, and consequently harm the DF performance to process more events per second. Since the multiple histogram bins (2 B per bin) can fit in one cache line (64 B), the memory access pattern might affect the number of cache misses. For example, if the detector electronics are sending randomly ordered events, every bin increment may yield a cache miss. On the contrary, if the detector electronics are sending sequentially ordered events by the channel number, then some bin increments will land in the same cache line limiting the number of cache misses. For this purpose, we perform our evaluations for 2 memory access patterns: *sequential* access where the events in the UDP packet are ordered by the channel number, and *random* access where is no specific order for the events.

Fig. 8 shows the event processing rate of both the DQDK and the Traditional OS Networking implementations as function of DQDK workers. For sequential memory access pattern, it is enough to employ 7 DQDK workers (14 CPU cores) to process the events of 9 TRISTAN tiles. However, we would need 8 DQDK workers (16 CPU cores) to perform the same task. For random memory access, 8 DQDK workers are needed for all setups. However, the DQDK implementation can process up to 165.88 million events per second which approximately corresponds to 10 TRISTAN tiles, while Traditional OS Networking can process up to 160.19 million events per second for random memory accesses and up to 159.77 million events per second for sequential memory accesses.

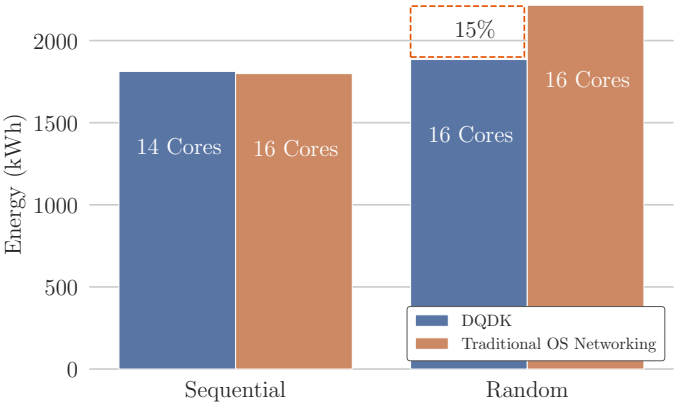

Fig. 9. Consumed Energy for 9-Tiles Histogram Construction

Fig. 9 shows the consumed energy for configurations that can process all events of the 9 tiles or more. For sequential memory access, we record the consumed energy of both: DQDK with the required 7 DQDK workers and Traditional OS Networking with 8 workers. Both configurations consume relatively the same amount of energy, but with our DQDK configuration is considered better because it can do the same with less CPU cores. For random memory access, 8 workers are required to process the events of 9 tiles both in DQDK and Traditional OS Networking and thus both configurations require 16 CPU cores. However, our DQDK implementation consume 330.7 kWh less (1884.55 kWh in total consumption) than Traditional OS Networking (2215.25 kWh in total consumption).

These results show that our DQDK implementation is superior to Traditional OS Networking in terms of performance and energy. The largest performance advantage of DQDK in TRISTAN appears when the processing latency of the software-based DF is relatively low, e.g. Waveform Mode. Even when the processing latency is higher and DQDK performance is close to Traditional OS Networking, e.g. in some cases in the Histogram Mode, the DQDK framework still saves up to 15% in energy.

IX. Conclusion

The Data acquisition Functions Virtualization (DFV) is a new data acquisition paradigm that minimizes the efforts to run, manage, and maintain software-based DAQ functions. DFV proposes to run software-based data acquisition functions on campus computing facilities that are already operated in most research institutes. The building blocks of DFV are high-performance computer networking and computer virtualization.

We presented the DQDK framework, a software library for high-performance and reliable DAQ. DQDK employs high-performance optimizations to realize the full power of AF_XDP and to prevent data loss caused by interference in campus computing facilities. The framework is scalable through parallelization by using more CPU cores. The user is responsible for the design and the performance of the logic supplied to the framework.

We apply DFV to the TRISTAN detector at KATRIN by simulating the detector operational modes. Our experiments show that we only require 6 CPU cores to process a 100 Gbit/s stream of TRISTAN detector in Waveform Mode and up to 16 cores for building a histogram in the Histogram Mode. The energy reductions are 10% in minimum and up to 15%. The performed experiments are executed in a virtualized environment using containers. This will allow the scientists to realize DFV and unleash its advantages for the TRISTAN detector. Using DFV, scientists can deploy software-based DF on campus computing facilities eliminating the efforts to run, manage, and maintain computer clusters on the TRISTAN experiment's site.

Acknowledgment

This work is supported by the Helmholtz Association and by the Ministry for Education and Research BMBF (grant numbers 05A23PMA, 05A23PX2, 05A23VK2 and 05A23WO6).

REFERENCES

[1] T. A. Bawej, U. Behrens, J. Branson, O. Chaze, S. Cittolin, G. L. Darlea, C. Deldicque, M. Dobson, A. Dupont, S. Erhan, A. K. Forrest, D. Gigi, F. Glege, G. Gomez Ceballos, R. Gomez-Reino Garrido, J. G. Hegeman, A. G. Holzner, L. Masetti, F. Meijers, E. Meschi, R. Mommsen, S. Morovic, V. O'Dell, L. Orsini, C. M. E. Paus, A. Petrucci, M. Pieri, A. Racz, H. Sakulin, C. Schwick, B. B. Stieger, K. Sumorok, J. Veverka, and P. Zejdl, "The New CMS DAQ System for Run 2 of the LHC," CERN, Geneva, Tech. Rep., 2015. [Online]. Available: https://cds.cern.ch/record/1711011

[2] J.-M. Andre, U. Behrens, J. Branson, P. Brummer, O. Chaze, S. Cittolin, C. Contescu, B. G. Craigs, G.-L. Darlea, C. Deldicque, Z. Demiragli, M. Dobson, N. Doualot, S. Erhan, J. F. Fulcher, D. Gigi, M. Gladki, F. Glege, G. Gomez-Ceballos, J. Hegeman, A. Holzner, M. Janulis, R. Jimenez-Estupinan, L. Masetti, F. Meijers, E. Meschi, R. K. Mommsen, S. Morovic, V. O'Dell, L. Orsini, C. Paus, P. Petrova, M. Pieri, A. Racz, T. Reis, H. Sakulin, C. Schwick, D. Simelevicius, and P. Zejdl, "The cms data acquisition - architectures for the phase-2 upgrade," *Journal of Physics: Conference Series*, vol. 898, no. 3, p. 032019, 10 2017. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/898/3/032019

[3] T. C. Collaboration, "The cms high level trigger," *The European Physical Journal C - Particles and Fields*, vol. 46, no. 3, pp. 605–667, 6 2006. [Online]. Available: https://doi.org/10.1140/epjc/s2006-02495-8

[4] P. Jenni, M. Nessi, M. Nordberg, and K. Smith, *ATLAS high-level trigger, data-acquisition and controls: Technical Design Report*, ser. Technical design report. ATLAS. Geneva: CERN, 2003. [Online]. Available: https://cds.cern.ch/record/616089

[5] S. Park, "Overview of the belle ii high-level trigger," in *The 2024 International Workshop on Future Tau Charm Facilities (FTCF2024)*, 2024.

[6] R. Sipos, "The ethernet readout of the dune daq system," *IEEE Transactions on Nuclear Science*, vol. 72, no. 3, pp. 317–324, 2025.

[7] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 164–177. [Online]. Available: https://doi.org/10.1145/945445.945462

[8] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in *2006 USENIX Annual Technical Conference (USENIX ATC 06)*. Boston, MA: USENIX Association, May 2006. [Online]. Available: https://www.usenix.org/conference/2006-usenix-annual-technical-conference/optimizing-network-virtualization-xen

[9] K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, C. Grandi, F. Grey, J. Harvey, A. Heiss, F. Hemmer, S. Jarp, R. Jones, D. Kelsey, J. Knobloch, M. Lamanna, H. Marten, P. Mato Vila, F. Ould-Saada, B. Panzer-Steindel, L. Perini, L. Robertson, Y. Schutz, U. Schwickerath, J. Shiers, and T. Wenaus, *LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005)*, ser. Technical design report. LCG. Geneva: CERN, 2005. [Online]. Available: https://cds.cern.ch/record/840543

[10] CERN Openlab, "Future ICT Challenges in Scientific Research," CERN, Tech. Rep., 2017, https://cds.cern.ch/record/2301895/files/Whitepaper_brochure_ONLINE.pdf.

[11] J. Albrecht, A. A. Alves, G. Amadio, G. Andronico, N. Anh-Ky, L. Aphecetche, J. Apostolakis, others, and T. H. S. Foundation, "A roadmap for hep software and computing r&d for the 2020s," *Computing and Software for Big Science*, vol. 3, no. 1, p. 7, 3 2019. [Online]. Available: https://doi.org/10.1007/s41781-018-0018-8

[12] Avolio, Giuseppe, Cadeddu, Mattia, and Hauser, Reiner, "Evaluating kubernetes as an orchestrator of the event filter computing farm of the trigger and data acquisition system of the atlas experiment at the large hadron collider," *EPJ Web Conf.*, vol. 214, p. 07024, 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921407024

[13] M. M. Storetvedt, "A new grid workflow for data analysis within the alice project using containers and modern cloud technologies," Ph.D. dissertation, Western Norway University of Applied Sciences, 2023, https://hdl.handle.net/11250/3061978.

[14] T. K. collaboration, M. Aker, K. Altenmüller *et al.*, "The design, construction, and commissioning of the katrin experiment," *Journal of Instrumentation*, vol. 16, no. 08, p. T08015, 8 2021. [Online]. Available: https://dx.doi.org/10.1088/1748-0221/16/08/T08015

[15] J. Li, M. Gu, F. Li, and K. Zhu, "An soa-based design of juno daq online software," *IEEE Transactions on Nuclear Science*, vol. 66, no. 7, pp. 1199–1203, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8674600

[16] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, p. 160018, 3 2016. [Online]. Available: https://doi.org/10.1038/sdata.2016.18

[17] M. Christensen and T. Richter, "Achieving reliable UDP transmission at 10 gb/s using BSD socket for data acquisition systems," *JINST*, vol. 15, no. 09, p. T09005, 9 2020. [Online]. Available: https://dx.doi.org/10.1088/1748-0221/15/09/T09005

[18] J. Mostafa, D. Tcherniakhovski, S. Chilingaryan, M. Balzer, A. Kopmann, and J. Becker, "100-gbit/s udp data acquisition on linux using af_xdp: The tristan detector," *IEEE Transactions on Nuclear Science*, vol. 72, no. 3, pp. 295–300, 2025.

[19] P. Emmerich, M. Pudelko, S. Bauer, S. Huber, T. Zwickl, and G. Carle, "User space network drivers," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. Washington, DC, USA: IEEE, 2019, pp. 1–12.

[20] M. Copik, K. Taranov, A. Calotoiu, and T. Hoefler, "rfaas: Enabling high performance serverless with rdma and leases," in *Proceedings of the 37th IEEE Interational Parallel and Distributed Processing Symposium*, ser. IPDPS '23, 2023.

[21] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff, "Revisiting the open vswitch dataplane ten years later," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 245–257. [Online]. Available: https://doi.org/10.1145/3452296.3472914

[22] W. A. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *SIGARCH Comput. Archit. News*, vol. 23, no. 1, p. 20–24, Mar. 1995. [Online]. Available: https://doi.org/10.1145/216585.216588

[23] S. Rusitschka, K. Eger, and C. Gerdes, "Smart grid data cloud: A model for utilizing cloud computing in the smart grid domain," in *2010 First IEEE International Conference on Smart Grid Communications*, 2010, pp. 483–488.

[24] A. Sanaee, V. Jabrayilov, I. Marinos, F. Shahinfar, D. Saxena, G. Antichi, and K. Kaffes, "Enabling fast networking in the public cloud," in *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 679–696. [Online]. Available: https://doi.org/10.1145/3779212.3790158

[25] W. Mansour, N. Janvier, and P. Fajardo, "Fpga implementation of rdma-based data acquisition system over 100-gb ethernet," *IEEE TNS*, vol. 66, no. 7, pp. 1138–1143, 2019.

[26] N. Schelten, F. Steinert, A. Schulte, and B. Stabernack, "A high-throughput, resource-efficient implementation of the rocev2 remote dma protocol for network-attached hardware accelerators," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 241–249.

[27] F. Zhang, N. Wang, Z. Hu, M. Wu, D. Wang, Z. Zhou, and Y. Wang, "A study of udp and tcp fpga implementation for data acquisition system," *JINST*, vol. 16, no. 07, p. P07044, 7 2021. [Online]. Available: https://dx.doi.org/10.1088/1748-0221/16/07/P07044

[28] J. Subraveti, "Implementation of UDP communication on a ZYNQ platform for processing clustered data based on multiple Gigabit Ethernet ports for phenoPET," Masterarbeit, Hochsch. Bremerhaven, Jülich, 2018, masterarbeit, Hochsch. Bremerhaven, 2017. [Online]. Available: https://juser.fz-juelich.de/record/844070

[29] M. Carminati, M. Gugiatti, D. Siegmann, K. Urban, P. King, F. Edzards, P. Lechner, S. Mertens, and C. Fiorini, "The tristan 166-pixel detector: Preliminary results with a planar setup," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1049, p. 168046, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168900223000360