



Discrete Event Modeling and Simulation-Driven Engineering for the ATLAS Data Acquisition Network

Matias Bonaventura, Daniel Foguelman, and Rodrigo Castro | Universidad de Buenos Aires, Argentina

Robust engineering methodologies offering product lifecycle control have proved to be a cornerstone in modern software development projects. Simultaneously, various modeling and simulation (M&S) techniques have become increasingly adopted in complex system design, particularly in scenarios in which it's difficult to predict system behavior as changes are introduced.

The DEVS (Discrete Event Systems Specification) framework is the most general formalism for modeling discrete event systems¹⁻³ and has been adopted in several disciplines for complex software and hardware system design and analysis.^{4,5} In addition to providing an unambiguous mathematical formalism to define model behavior and structure, DEVS

provides a clear framework for system analysis, experimental frame definition, model-to-simulator verification, and model-to-system validation.

We present a DEVS-based methodology for M&S-driven engineering projects that integrates software development best practices tailored to a large-scale networked data acquisition system in a physics experiment (specifically, the ATLAS particle detector⁶ at CERN⁷). This project poses M&S challenges from several viewpoints, including system complexity, tight delivery times, the quality and flexibility of the developed models and tools, interdisciplinary communication of results to collaborators (mostly scientists), and big data-scale analysis.



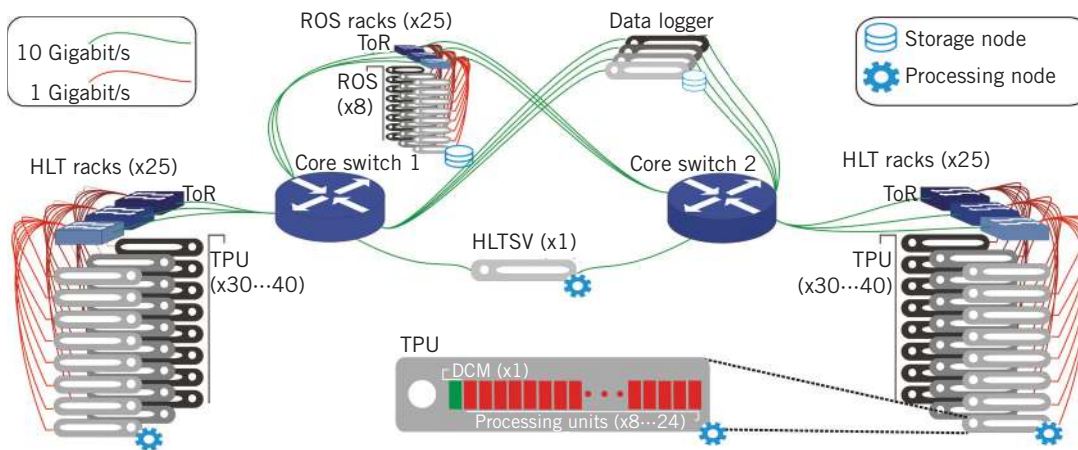


Figure 1. Topology and applications in the high-level trigger and data acquisition (TDAQ) farm. This intermediate configuration is from long shutdown 1 (LS1) in 2014.

The Data Acquisition Network at CERN's ATLAS Experiment

The Large Hadron Collider (LHC)⁸ is the world's largest particle accelerator—27 kilometers in circumference—colliding bunches of particles (protons or ions) every 25 ns near large detectors, including ATLAS, CMS,⁹ ALICE,¹⁰ and LHCb.¹¹ In 2013, the Run1 detectors went offline for maintenance and upgrades (long shutdown 1, or LS1) until the Run2 restart in 2015. Collisions in the ATLAS detector generate very high energy, enabling the search of novel physical evidence such as Higgs boson, extra dimensions, and dark matter. Each particle bunch collision is called an *Event* (we use “Event” for high-energy physics and “event” for DEVS modeling) and consists of particle-induced signals registered in the detector and digitized for further analysis. The raw amount of information generated exceeds 60 Terabyte/s.

To assimilate this throughput, ATLAS uses a sophisticated layered filtering system (trigger and data acquisition, or TDAQ¹²) that decides in real time whether each Event should be permanently stored or safely discarded. The first-level trigger (L1) filters Events from an initial raw rate of 40 million Events/s down to a filtered rate of 100,000 Events/s. L1-accepted Events are temporarily stored in a read-out system (ROS) in the form of data structures called *fragments* and then accessed by a second-level filter called the high-level trigger (HLT). At the HLT, physics algorithms reanalyze the fragments (this time with a different granularity), retaining only 1,000 “interesting” Events/s. The TDAQ system and its HLT-ROS data network is our system under study.

Applications and Data Network in the HLT

Figure 1 shows the interconnections among various applications in the HLT at the commencement of our case study. Upon selection by L1, Event data is transferred to the ROS, and the specialized application HLT supervisor (HLTSV) is notified. The HLTSV assigns Events to trigger processing unit (TPU) servers, which run an application called a data collection manager (DCM) to centralize communication between the TPU and the rest of the system. DCMs interface with instances of the application processing unit (PU)—one per available core, between 8 and 24 per host. Each Event is assigned to a single PU instance that analyzes it and decides whether it should be permanently stored or discarded. This system represents our starting point for the M&S process.

Applications communicate over an Ethernet network with link capacities of 1 and 10 Gbps. Two core routers and approximately 100 switches interconnect roughly 2,000 multicore servers using TCP/IP protocols. Figure 1 shows a diagram of the network. The farm is composed of 50 racks for TPU servers and 25 racks for ROS nodes. Each TPU rack contains from 30 to 40 servers (DCMs and PU applications), and each ROS rack contains 8 servers. Within each rack, servers are connected to a shared top-of-rack (ToR) switch via 1 Gbps links. The HLTSV node and the ToRs are connected to the core switches over 10 Gbps links.

DEVS for Data Network Modeling

DEVS is a mathematical formalism for M&S based on general systems theory—that is, it's independent of any specific application. DEVS lets us describe

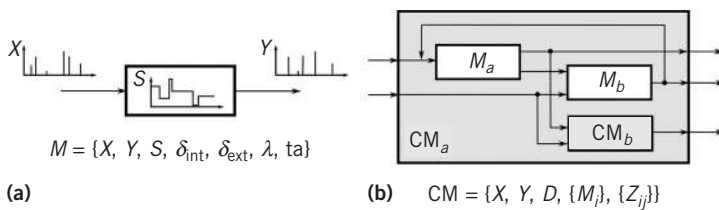


Figure 2. Basic Discrete Event Systems Specification (DEVS) (a) atomic models and (b) coupled models. Coupled models define the structure of the system (interconnections between coupled and atomic models). Atomic models define the dynamic behaviors.

exactly any discrete system and approximate numerically continuous systems with any degree of desired accuracy. The formal model specification provides tools for analytical manipulation and offers independence in choosing the programming language for implementation.² DEVS models are described as a hierarchical composition of atomic models (M_s) and coupled models (CMs) defined by mathematical tuples as shown in Figure 2.

CMs define system structure (interconnections between coupled and atomic models), whereas M_s define dynamic behaviors. For M_s , each possible model state $s \in S$ has an associated lifetime defined by the function $ta: S \rightarrow R_0^+$. When the model is in state $s = s_1$, at time $t_1 = ta(s_1)$ it autonomously undergoes an internal transition toward a new state $s_2 = \delta_{int}(s_1)$, where $\delta_{int}: S \rightarrow S$ is the *internal transition function*. An output event is simultaneously produced at t_1 with value $y_1 = \lambda(s_1)$, where $\lambda: S \rightarrow Y$ is the *output function*.

When a model receives an input event $x_1 \in X$, an external transition is triggered that instantly changes the model state to $s_4 = \delta_{ext}(s_3, e, x_1)$, where s_3 is the model state by the time it receives the input event, and e is the elapsed time since the last state transition (with $e < ta(s_3)$). The function $\delta_{ext}: S \times R_0^+ \times X \rightarrow S$ is the *external transition function*.

Vectorial DEVS. The DEVS simulation algorithm is universal, unambiguous, easy to implement, and independent of programming languages, with many of its extensions and specializations tackling different needs. We're particularly interested in vectorial DEVS (VDEVS),¹³ which lets us model large-scale systems with a compact graphical representation. A vectorial model is an array of quasi-identical classic DEVS models that can differ in their initial parameters. Formally, the vector model's structure is defined by $VD = \{N, X_v, Y_v, P, M_i\}$, where N is the vector dimension, X_v is the set of input events vector, Y_v is a vector set of output events, P is the set of

parameters, and each M_i is a classic DEVS model. For the interaction between vectorial and nonvectorial, we define scalar to/from mappings of vector models.

PowerDEVS. We developed a model for TDAQ using the PowerDEVS tool,¹⁴ which provides a graphical interface to define DEVS models via block diagrams, a C++ editor to code the four dynamic functions for the M tuple, and libraries with reusable models. PowerDEVS also has a native interface to Scilab (www.scilab.org), an open source alternative to Matlab for numerical computation purposes. We adopted a data networks library (queues, servers, traffic generators, a TCP implementation, and so on^{15, 16}) and extended it for our case study.

Network-specific simulators strive to represent protocols and hardware nodes in great detail. They typically provide comprehensive and reusable libraries that allow for quick model prototyping—for example, OMNeT++ (www.omnetpp.org), NS2/3,¹⁷ and OPNET¹⁸ (an updated review¹⁹ and a recent simulation study²⁰ of the TDAQ system using OMNeT++ appear elsewhere).

When adopting prebuilt network frameworks, it's difficult (or even impossible, depending on the software package) to freely choose the desired simulation abstraction level. Experience shows that once a question is defined, several protocol features (or even entire network layers) can become dispensable as they don't contribute significantly to increase result fidelity, but they do increase simulation costs.²¹ This poses risks in M&S projects, particularly for large-scale networks.

By adopting a general-purpose discrete event formalism such as DEVS, we partially renounce some out-of-the-box detailed protocol features offered by network-specific packages, but we gain the freedom to decide what kind of representation and granularity suits a given stage of the project. Our strategy for modeling the TDAQ system is to flexibly select a sufficient level of abstraction to answer each particular question with an acceptable fidelity given time and computational resource constraints. Along these lines, we aim to perform hybrid simulations (discrete events mixed with continuous flows). This capability is readily available in DEVS²² and implemented in advanced versions of PowerDEVS tailored for data networks.¹⁶

Context, Requirements, and Methodology

For any case study that might arise in TDAQ, cross-cutting contexts and requirements call for a flexible yet robust development methodology.

The TDAQ HLT filtering farm is no exception. During LS1, it was subject to hardware and

Table 1. Elicited requirements.

Requirement	Goal
Evaluate candidate changes for the network and control algorithms before their commissioning	Perform early risk assessment
Define in advance the best set of tests to perform on the real system during scarce windows of availability	Harness the test window to focus on the most relevant questions
Enable flexibility for choosing the level of detail/accuracy with which the evaluations are obtained	Dynamically adapt to different and complex modifications that need to be assessed, and then schedule changes

control algorithm changes that affect network topology and throughput, yet predicting the impact these changes have isn't straightforward. Serious design and benchmark studies on system components give confidence, but they require access to the hardware in advance. In the end, testing the system as a whole happens only at the final integration phase.

The full TDAQ system was available for testing only about one out of every six weeks (during scheduled technical runs), which delays testing on new control algorithms that are continuously improved but can't be fully validated until the full system is available.

Table 1 lists the resulting requirements elicited during system analysis meetings. Moreover, these requirements are likely to change dynamically throughout a project's lifetime, with different experts having varying requirements on the same system component's analysis.

To implement an engineering strategy driven by modeling and simulation, we proposed the iterative process-based methodology illustrated in Figure 3.

DEVS Formal Framework

At the methodology's core, the system, model and simulator entities are strictly separated yet formally related by the DEVS framework. The real (or "source") system is experimented under a system experimental frame (EF_S), with questions encoded in the form of system parameters Θ_S that define experimental conditions. Experimental results relevant to the original questions are stored in a system behavior database λ_S .

As a specification of structures and behaviors, every new DEVS model is built for a pair {System, EF_S} according to a modeling relation and guided by selected homomorphisms/isomorphisms. A new model experimental frame (EF_M) also allows for questions about model attributes (using model parameters Θ_M for queries and a model database

λ_M to store answers) related to coupling density, model topology, types of variables (discrete, continuous), and so on, with no access to the real system and independent of any simulation exercise.

A DEVS simulator reads a DEVS model and produces an output trajectory by obeying the model's dynamics (in short, a DEVS model is simulated). Its most common realization is a computer program, usually referred to simply as a simulator, which is constructed, adapted, and maintained to read and compute DEVS models efficiently within their EF_M. This establishes a simulation relation. The compute experimental frame (EF_C) defines new questions and parameters Θ_C for experimenting with (simulating) the computable model. It also hosts simulation results in a compute behavior database λ_C . The validation relationship lets us relate back to the original system to validate correctness (λ_S versus λ_C) or to perform scans over EF_S due to unexpected observations discovered in the EF_C.

Cycles and Phases

We organize the flow of tasks in three main cycles: *build* (the model) in blue, *hypothesis* (on the system) in orange, and *explore* (simulation results) in green. While each cycle's goal differs, in all cases the flow across the DEVS formal framework follows the system → model → simulation path. In turn, for each evolution through the cycle, two parallel and cooperative phases are defined: the *system study* phase drives progress according to questions about the system under study, and the *tools development* phase seeks to improve the supporting software algorithms and interfaces, leveraging modeling, simulation, and analysis capabilities.

The build cycle starts with observation and measurement of the system. Its objective is to provide quality models that, once simulated, will exhibit an adequate degree of validation against the original system. The hypothesis cycle exercises on the model several

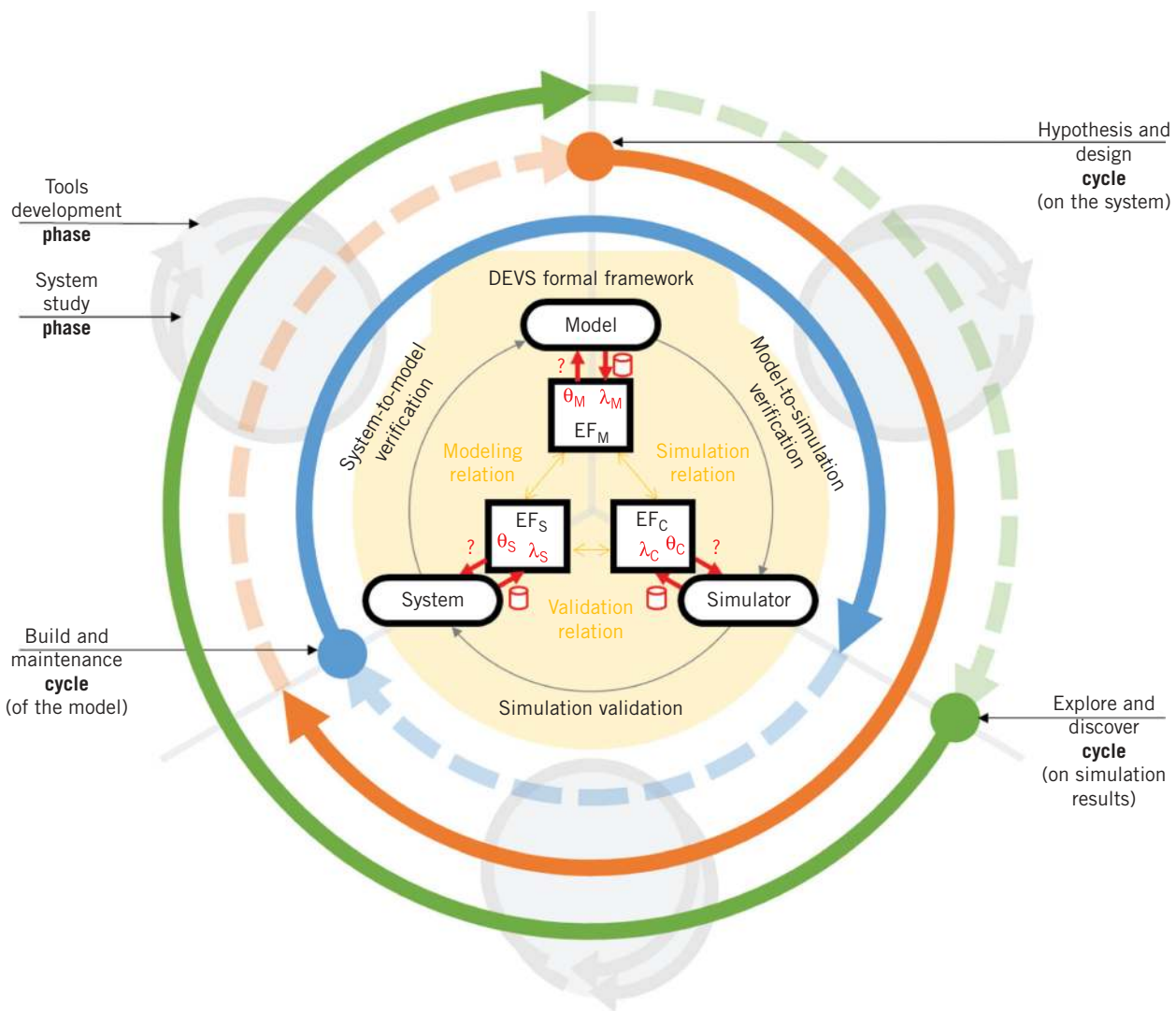


Figure 3. Modeling and simulation-driven engineering. The methodology diagram based on the DEVS formal framework shows iterative cycles and incremental phases.

candidate changes to be applied onto the system. Its goal is to find improvement opportunities for the system when it's unavailable or when direct experimentation is too expensive. The explore cycle starts with analyzing the large amounts of information produced by simulations; its goal is to discover properties and correlations unthought of during the experimentation phases.

Cycles need not occur in any specific order (although a build cycle is usually required at the beginning of a project). This approach leads to a model that reproduces relevant behaviors of the real system within reasonable simulation times: less relevant dynamics are kept out of the model (such as intrinsic of the network physical layer). The methodology also offers a guideline for development phases of the

underlying modeling and simulation software tools; new features are added to the tools at specific phases, responding to specific needs, framed within unambiguous cycle goals.

Existing Techniques and Methods

Software engineering processes and methodologies propose frameworks to control software projects' life cycles—some of the most popular are test-driven development, extreme programming, and the Rational Unified Process. Some of these foster practices such as pair programming or code reviews as part of this work, whereas others propose iterative and incremental cycles, with frequent deliveries focused on adding value quickly.

Our methodology shares some aspects with these approaches. However, none of the aforementioned methods include the formal M&S aspects provided by DEVS: strict separation between modeling formalism, abstract simulation mechanism, and code implementation (of both model behavior and simulation engines). This gives the advantage of independence between experimental frames for the real system, the model, and the simulator, straightforwardly propagating enhancements in any of these three areas to the others. In typical software-based projects, it's unusual to modify the base tools themselves to execute the project. However, in M&S-driven scientific projects, the base tools for modeling, simulation, and data analysis are crucial devices that call for their own requirements alongside requirements of the model itself. Our methodology naturally fills this need.

Large sets of simulation results can support data-driven hypothesis and predictive analytics.²³ A well-structured simulation database together with reusable data analysis libraries can systematize different layers of information aggregation, enabling stratified levels of analyses. Our methodology fosters this approach.

Case Study: Improving the TDAQ Flow and Data Network

Our real-life case study, in which we applied the above presented methodology, starts with two build cycles, observing the system, translating knowledge into an executable simulation model, and upgrading the model to represent important design changes in the system.

This step was followed by an explore cycle, in which we discovered hidden undesirable behaviors in load balancing mechanisms. Such behaviors were confirmed to exist in the real system and raised the need for improvements along with open questions about possible solutions.

To answer the new questions and provide for predictions, we used hypothesis cycles to test alternative scenarios that weren't rapidly exercisable on the real system. We then implemented into the real application a set of improvements that proved satisfactory in the simulated environment. Finally, we evaluated their true effectiveness in the real network by loading the system with emulated physics Events.

The model focuses on predicting HLT dataflow performance. We selected filtering latency as the main performance metric; it represents the time from when the HLTSV assigns an Event to a given PU until when the Event is either discarded or stored.

The sequence diagram in Figure 4 depicts the applications that take part in Event filtering. The PUs request information from the ROS in two stages:

L2 filtering and Event building (EB). In L2, a small portion of the Event is first requested and then analyzed; this step can be repeated several times until EB takes place and all pending information is requested as a whole. For each requested portion of the Event, all involved ROS nodes send their replies to the same DCM almost simultaneously, creating traffic bursts from ROS → DCM that increase the filtering latency because of the queuing effect generated at the core and ToR switches.

TDAQ has high bandwidth and low latency in relation to TCP minimum retransmission time (200 ms). Together with the data flow described earlier, these conditions create a TCP throughput collapse known as the TCP Incast pathology.²⁴ The impact on TDAQ can be huge. Whenever a single TCP packet is discarded at the switches, a PU can't start processing the Event until that packet is retransmitted (after 200 ms at best), raising the perceived network latency of an Event request from a theoretical minimum of 19.2 ms (for 2,400 bytes) to more than 200 ms. To avoid the Incast effect, the DCM application restricts the number of simultaneous requests to the ROS using a credit-based traffic shaping control that limits "in flight" requests on the network.²⁵ Because responses can vary significantly in their size, traffic shaping doesn't completely prevent packet losses, so it's important to study the effects of queue saturation (and TCP retransmissions) and engineer the network and its algorithms to maximize performance and minimize high-latency risks. This is where our M&S-driven network engineering methodology comes into play.

First Iteration: Building the Model

We start the model implementation with a build cycle (blue cycle in Figure 1). We defined the system experimental frame EF_S for this cycle as a subset of the complete system: the HLTSV, all ROS nodes, and a single instance of the DCM and PU applications. To simplify timing calculations we assumed zero processing time at the PUs, and Events with fixed size (2.4 Mbytes). This EF_S is representative of the entire system with unlimited resources, as each PU independently processes a single Event at a time. Scaling this scenario shows emergent behaviors of resource sharing (DCM credits, network bandwidth, and so on).

Real system measurements. The build cycle begins with observation of the real system (experimentation and metrics acquisition), so we measured filtering latency in different scenarios. Experiments were defined using $\Theta_S = \{\text{number of initial DCM credits}\}$

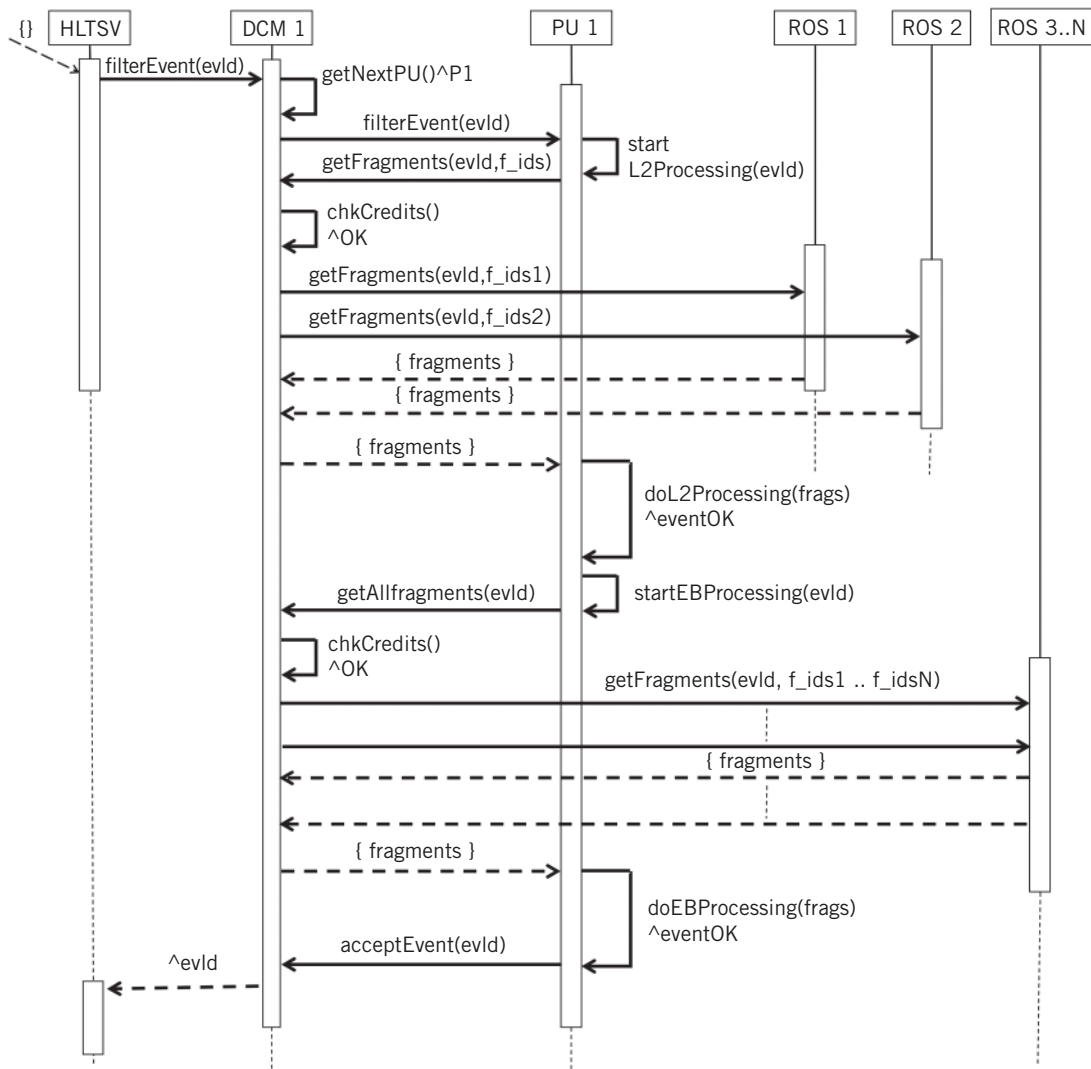


Figure 4. TDAQ application sequence diagram involved in filtering a single Event. The processing units (PUs) request information from the read-out system (ROS) in two stages: level-two (L2) filtering and Event building (EB).

and results stored in λ_5 . In Figure 5a, we see an optimum configuration in which average latency stabilizes at 20 ms (close to the theoretical minimum) within a range of about 100 to 600 DCM credits. With fewer credits (12 to 100), latency increases (DCM can send fewer simultaneous requests, underutilizing network capacity). Using more than 600 credits, latency increases rapidly and stabilizes at around 500 ms. We observed packet discards on the ToR switches when more than 600 credits were used, thus confirming that the latency increase is due to network congestion and TCP retransmission (no packet loss was observed at core switches).

Model implementation. The build cycle continues with the creation of a DEVS model guided by the

TDAQ architecture and data flow described for a single PU application. Figure 6 shows a PowerDEVS view of the implemented TDAQ model.

To preserve the real system's semantics, we built a hierarchical model complying with TDAQ naming and structure conventions. This greatly facilitated the extraction of control logic from the C++ algorithms in the real applications, thus maximizing the homomorphism with the system under study. The ROS and DCM coupled models implement the TCP flow and congestion control logic based on preexisting PowerDEVS libraries. TCPsSender models TCP Cubic,²⁶ implementing only the TCP behavior relevant to the case study. Tests to validate the TCP model against the real system shifted our focus from the average latency (red curve in Figure 5) to

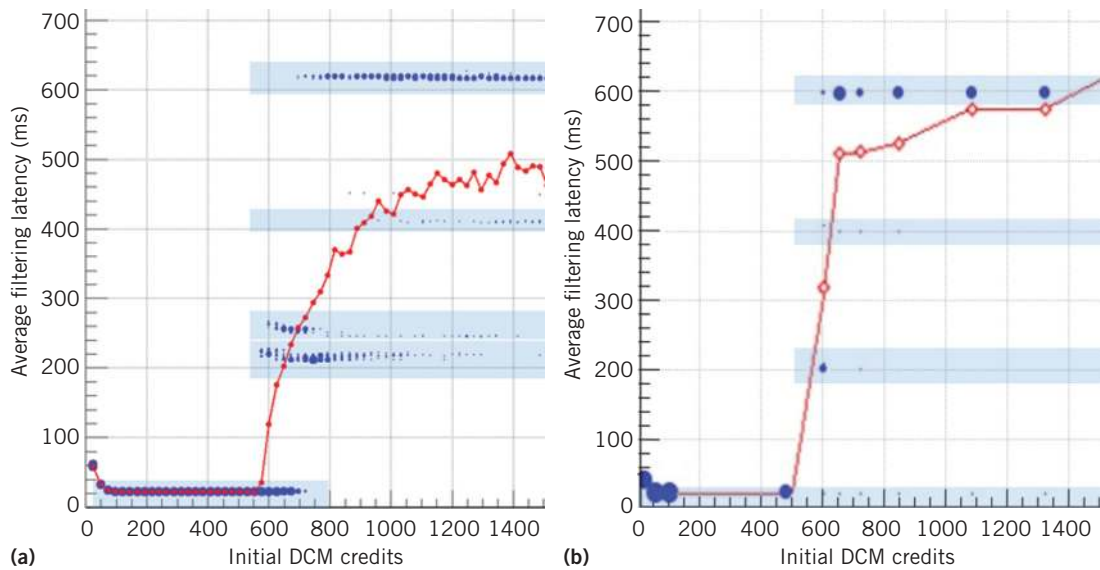


Figure 5. Filtering latency versus initial DCM credits: (a) real system measurements and (b) simulation results. The red curve shows average latency, and blue dots show individual latencies; larger dot clusters denote higher number of occurrences, which gather around discrete ranges (close to 15 ms, 200 ms, 400 ms, and 600 ms).

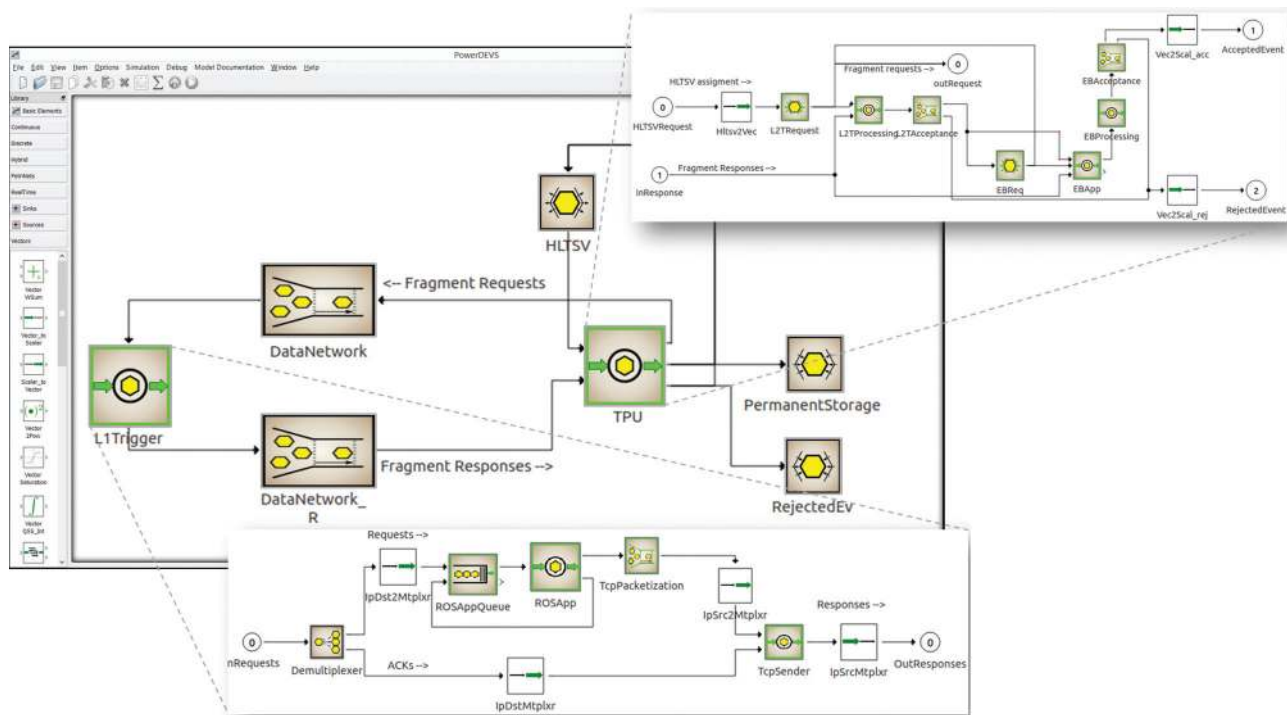


Figure 6. TDAQ simulation model implemented in PowerDEVS. Tests to validate the TCP model against the real system shifted the focus from studying averaged filtering latencies to analyzing clustered latency patterns (red curve vs. blue dots in Figure 5).

the clustered latencies pattern (blue dots). While the explanation for the occurrence of clustered latencies is outside this article's scope, it has a central role in the TCP Incast effect. Moreover, the modeling ef-

forts led to the detection of a bug in the Linux SCL6 TCP implementation that's responsible for the (unexpected) cluster around 600 ms (https://bugzilla.redhat.com/show_bug.cgi?id=1203742).

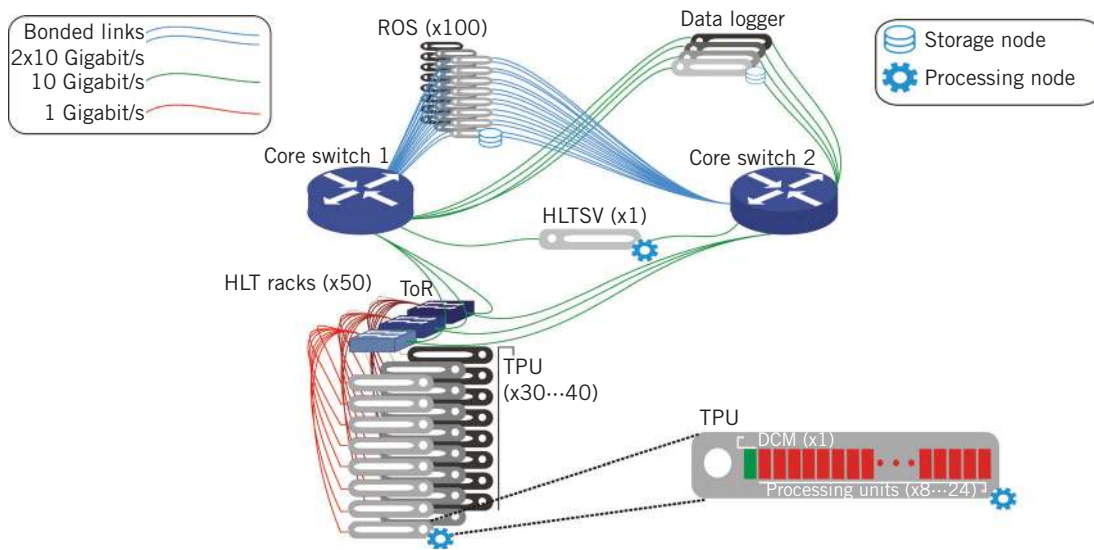


Figure 7. Topology and applications in the TDAQ HLT farm for Run2. This is an upgrade of the one in Figure 1.

Following the tools development approach, we implemented TCP atomic models (sender and receiver) and network elements (channels and switches) to be generic and reusable and incorporated them into the PowerDEVS network library. We also implemented new Scilab and ROOT²⁷ visualization mechanisms for latency post-analysis along with a new distributed simulation infrastructure, allowing us to execute multiple simulations for parameter sweeping purposes. These tools are meant to be reused in generalized simulation applications.

Simulation results and validation against the real system. The next step of the build cycle is model verification and simulation validation. We configured the simulation to follow the real system setup described earlier (controlled $\Theta_S \rightarrow \Theta_M \rightarrow \Theta_C$ translation), sweeping the number of initial DCM credits. Figure 5b shows the results. The simulation reproduces the individual filtering latencies (blue dots) following the same clustered patterns, validating the TCP dynamics (retransmissions and TCP Incast effect). The simulated average latency approximates real measured latencies ($\lambda_S \sim \lambda_C$), with 100 to 600 credits attaining minimum latency and fewer than 100 credits slightly increasing latency. For credits above 600, the simulation showed congestion and packet drops on the ToR switches, but the increase in average latency was much steeper compared to the real system. Another difference was the stabilization point under congestion: the real system latency stabilizes at 500 ms, whereas the simulated latency grows up to 700 ms. Although these differences require further study, the simulation reproduces

very closely the intervals of major interest, underlining the constant tradeoff among degrees of model detail, simulation accuracy, and delivery times for a given engineering concern.

An important advantage of the simulated model is that it allows for fine-grained analysis (packet by packet if required). For example, link utilization and queue occupancies can be visualized and studied in detail in the simulation, but it's impossible to sample the instantaneous evolution of queue occupancies at network devices (for example, to pinpoint queuing bursts that are critical for TDAQ and occur in less than 8 ms).

Second Iteration: System Upgrade and Model Improvements

In the second iteration of the build cycle, we expand the system's experimental frame EF_S by increasing the number of TPUs and of PU applications on each TPU. During this cycle, the real system was upgraded, calling for changes in the model.

Changes in network topology. The TDAQ team commissioned several changes in the HLT network in preparation for ATLAS's Run2 phase, which doubles the maximum particle's collision energy. The ROS ToR switches were removed and the 200 ROS nodes replaced by 100 new computers with four 10 Gbps interfaces, each directly connected to both core switches. The ToR switches were expanded with additional 10 Gbps links to both core switches. The overall throughput supported at the network level increased by one order of magnitude (see Figure 7).²⁸

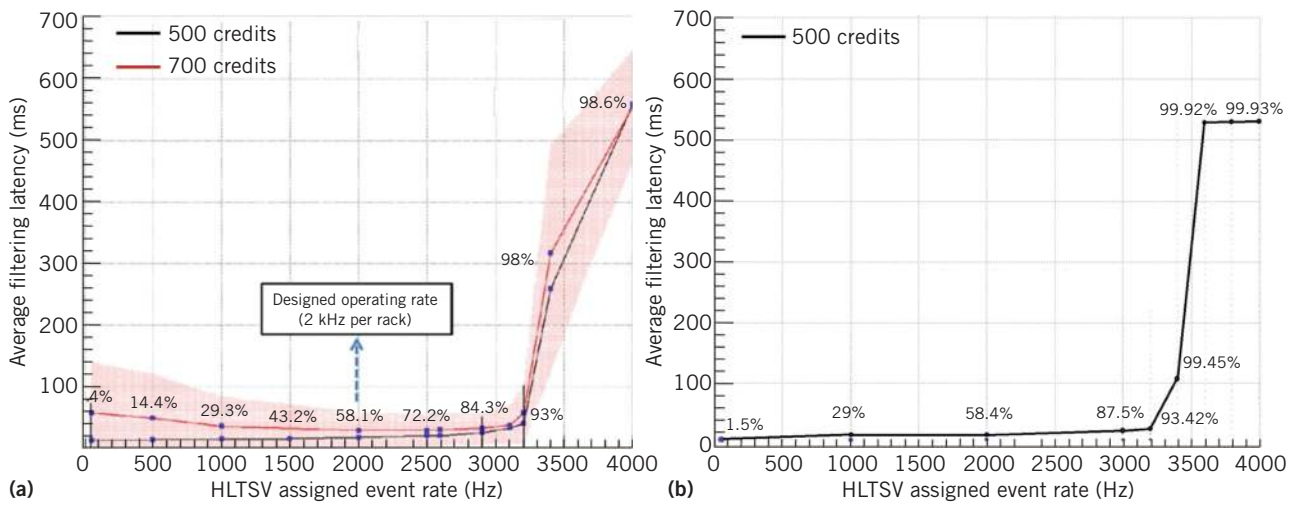


Figure 8. Average Event latency sweeping the HLTSV assignment rate (200 ROS, 1 TPU rack with 40 DCMs, 960 PUs): (a) real system measurements and (b) simulation results. Percentages represent network load, and red background shows standard deviation.

Real system measurements. Again, the first step in the build cycle is taking real metrics from the upgraded system (all new ROS nodes and a full rack of TPUs), where the network traffic is largely determined via HLTSV assignment rate. With a 100 kHz rate for the HLTSV and 50 TPU racks (full farm), each rack should handle Events at 2 kHz. Thus, new experiments must sweep this parameter ($\Theta_S = \{\text{HLTSV rate}\}$), ranging from 50 Hz (nonsharing of resources; Events are processed faster than 20 ms) up to 4 kHz (network saturation point). To simplify the analysis, we used a synthetic configuration: PUs accept Events 50 percent of the time, Event size is 1.3 Mbytes, and the DCM uses 500 and 700 credits.

Figure 8a shows the average Event latency for increasing the HLTSV assignment rate. When the HLTSV assigns Events at 50 Hz, latency is minimal (13 ms) because the network is completely free when applications start filtering Events. For increasing assignment rates, latency rises as several PUs simultaneously request Events competing for finite network resources and DCM credits. For rates above around 3.2 kHz, latency increases exponentially as the network approaches its maximum capacity (93 percent utilization).

Model implementation. Model changes related to topology upgrades were minimal: the ROS ToR switch models were easily removed, thanks to the modularity fostered by DEVS, and the channel's configuration changed to match the new link capacities. This shows the model's flexibility and the advantage of having a one-to-one mapping between components of the real system and the simulation model. At this

stage, we developed a complete HLTSV implementation, reusing directly some chunks of C++ code from the real HLTSV application for greater reliability. To increase the number of model instances, we used VDEVS, developing 16 new vectorized DEVS models and 10 new multiplexer models to represent packet routing.

For the tools development phases, we implemented three generic solutions to address the scalability requirement of increasing the number of simulated instances 50 times. VDEVS's original proposal was extended, allowing for C++11 SmartPointers in vector DEVS messages. SmartPointers were also included directly in the PowerDEVS simulation base engine to allow for automatic and transparent memory management in any atomic DEVS model. This approach dramatically reduced the simulator's memory footprint, pushing its scalability to the next order of magnitude. We also developed a new general framework for PowerDEVS to automatically launch simultaneous simulations on distributed nodes, reducing simulation times for parameter sweeping experiments linearly with respect to the number of nodes used (simulations are completely independent).

Simulation results and validation with the real system. To complete the build cycle, we validated the simulations against the real system, replicating previously conducted experiments to sweep the HLTSV assignment rate parameter. We executed nine experiments, each simulating 60 seconds (180,000 filtered Events in the most stringent case) in three different nodes, completing all simulations in 120 minutes. As Figure 8b shows,

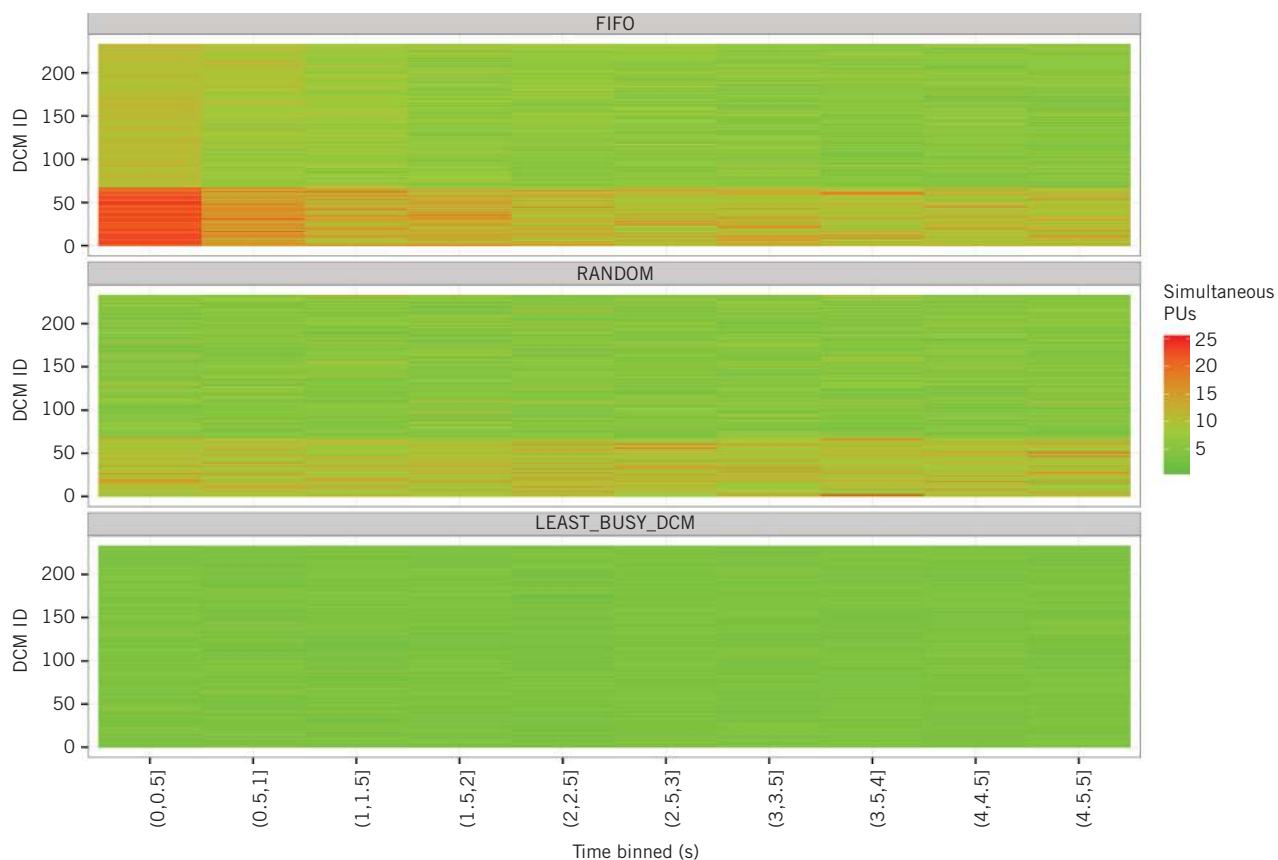


Figure 9. Heatmap of the load in the HLT farm for different HLTSV assignment policies. Tile color represents the maximum amount of PUs simultaneously processed in each DCM (230 DCM IDs in the vertical axis) in 0.5 s (5 s binned in the horizontal axis).

the simulation results closely reproduce the latency curve measured in the real system. The absolute latency values and network load on the simulation differ from reality within an acceptable range (less than 5 percent difference), showing a good degree of validation.

Third Iteration: Exploring and Discovering System-Level Behaviors

The simulation accuracy obtained in previous cycles provides us with a sufficient confidence level on the simulated data that justifies running an explore cycle (green cycle in Figure 3) to find potential emergent behaviors.

Full system simulations generate huge volumes of information for the 6 million Events filtered in a single minute (processing times, filtering latencies, queues occupancy, link usages, farm utilization, and so on). Some of this information isn't available in the real system or is too difficult to gather uniformly for post-analysis goals.

Figure 9 is an example data analysis performed on the simulation results for λ_C . It shows how Events

are distributed across the farm in different time slots using various load-balancing algorithms: first in first out (FIFO) is the default policy implemented for selecting the TPU node that will filter the next Event.

In the FIFO policy, the reddish area at the bottom explains the fact that 30 percent of the DCMs had double the amount of PUs available for processing, thus explaining their higher load. All DCMs are heavily assigned in the first time bins; after a few seconds of execution, load becomes similar to the RANDOM algorithm (each Event's filtering time differs). Another detected system-level behavior is that individual DCMs differ significantly in the number of Events they process—the color intensities vary noticeably along any single row and along any single column. These observations led us to infer that a potentially uneven load-balancing mechanism might be the cause of overall higher filtering latencies.

For the tools development phase, we developed a set of reusable R libraries for data analysis and visualization of the large volumes of logging information produced by PowerDEVS. The new graphical

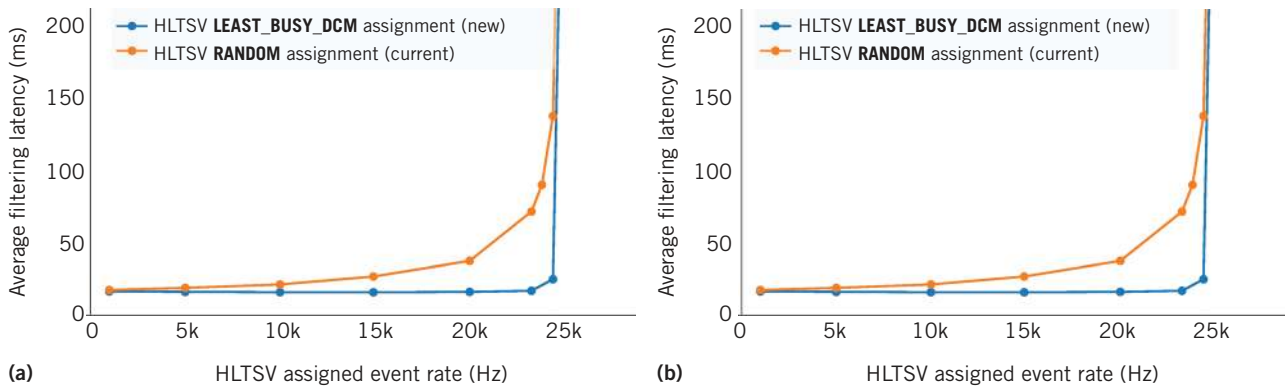


Figure 10. Comparison of assignment policies (RANDOM versus LEAST_BUSY_DCM) for (a) real system measurements and (b) simulation results. The RANDOM algorithm exhibits the same behavior as the FIFO algorithm, while the new algorithm maintains average Event latency close to a minimum (16 ms) for all frequencies below 24 kHz.

information generated through the R platform (such as the heatmaps in Figure 9) became standard means of communication with the TDAQ team.

Fourth Iteration: Real System Improvement Proposal

The revealed behavior discovered during the explore cycle moved us forward to the hypotheses cycle (orange cycle in Figure 3) to test new load-balancing algorithms in a simulated domain in search of improved performance.

Testing the hypothesis on the model. The latency's linear increase in Figure 8 is the effect of several PUs competing for the same resources (reddish tiles in Figure 9). However, in our synthetic experiment for the designed frequency (2 kHz per rack), each DCM receives on average one assignment every 50 to 60 ms, while the minimum latency for filtering an Event is roughly one third of this period (with an unloaded network). Under these conditions and an optimal assignment policy, it isn't necessary for two PUs to process simultaneously on a single DCM. However, such assignments currently behave as random (uniformly distributed), so sometimes 10 to 25 PUs of the same DCM process simultaneously while other DCMs are almost idle (DCM load in Figure 9).

We modeled three HLTSV assignment policies: FIFO, used by the real system; RANDOM, in which the HLTSV selects a random idle PU; and the new LEAST_BUSY_DCM, in which the HLTSV selects an idle PU within the DCM with fewer busy PUs. The main idea behind LEAST_BUSY_DCM is to revert the uneven load detected in the explore cycle by assigning Events according to the load on each DCM.

Simulation results. After implementing new alternatives in the model, we performed simulations to compare the RANDOM algorithm with the proposed LEAST_BUSY_DCM algorithm (FIFO is omitted because it eventually becomes equivalent to RANDOM as shown in Figure 9). To compare, we simulated the same experiment as in the second iteration (sweeping the HLTSV rate) but configured nine TPU racks (267 DCMs and 6,408 PUs). Figure 9 shows that LEAST_BUSY_DCM effectively balances the load of all DCMs in the farm, reducing the amount of simultaneous PUs processing in each DCM (tile colors present more similarity along rows and columns). Figure 10b shows simulation results comparing both algorithms. The RANDOM algorithm exhibits the same behavior as the FIFO algorithm, while the new algorithm maintains average Event latency close to a minimum (16 ms) for all frequencies below 24 kHz. For higher frequencies, the latency grows exponentially due to network congestion. These results suggest that the new algorithm could reduce latency between two to four times for this specific configuration (design rate of 15 kHz with a network saturation point of 23 kHz). New tests are under way with more realistic data flow to increase validation confidence.

Implementation and validation in the real system. Once we test the hypothesis in the simulation, the next step in the hypothesis cycle is to implement changes to validate against the real system. It was possible to reuse some C++ code developed for models in the simulation, with minor adaptations to attain close-to-real-time performance (the 100 kHz rate requirement for HLTSV is a stringent one). We

performed the same experiment in the real system, with HLTSV rate sweeping using nine TPUs racks. Figure 10a shows the result of comparing RANDOM and LEAST_BUSY_DCM algorithms in the real system. With the new algorithm and rates under 24 kHz, the average latency is kept to a minimum and shows improvements of two to four times compared to the current FIFO algorithm, as predicted in the simulation. The simulation is thus validated, showing that the model is capable of reproducing known behaviors, representing a valuable tool to predict the impact of changes in the real system.

We're currently implementing our model with a variety of TDAQ scenarios in which we study different candidate traffic control techniques in search of further performance improvements (in particular, looking for quick recovery times in the face of system failures). We also plan to apply our methodology and tools to assess candidate ATLAS upgrades (planned for 2018), comparing performance and modeling techniques with other simulation frameworks. Ongoing research aims to automate parameterization-simulation-validation cycles by retrieving real run parameters and metrics recorded in the ATLAS Information Service database. ■

References

1. B.P. Zeigler, *Theory of Modeling and Simulation*, John Wiley & Sons, 1976.
2. A.C.H. Chow and B.P. Zeigler, "Parallel DEVS: A Parallel, Hierarchical, Modular, Modeling Formalism," *Proc. 26th Conf. Winter Simulation*, 1994, pp. 716–722.
3. B.P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000.
4. G. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, CRC Press, 2009.
5. G. Wainer and J. Mosterman, *Discrete-Event Modeling and Simulation: Theory and Applications*, CRC Press, 2010.
6. ATLAS Collaboration, "The ATLAS Experiment at the CERN Large Hadron Collider," *J. Instrumentation*, vol. 3, no. 8, 2008, p. S08003.
7. D. Pestre, "L'organisation Européenne pour la Recherche Nucléaire (CERN): A Succès et Politique Scientifique," *Vingtième Siècle, Revue d'Histoire*, JSTOR, 1984, pp. 65–76.
8. L. Evans and P. Bryant, eds., "LHC Machine," *J. Instrumentation*, vol. 3, no. 8, 2008, p. S08001.
9. CMS Collaboration, "The CMS Experiment at the CERN LHC," *J. Instrumentation*, vol. 3, no. 4, 2008, pp. 1748–0221.
10. K. Aamodt et al. (ALICE Collaboration), "The ALICE Experiment at the CERN LHC," *J. Instrumentation*, vol. 3, no. 8, 2008, p. S08002.
11. A.A. Alves Jr. et al. (LHCb Collaboration), "The LHCb Detector at the LHC," *J. Instrumentation*, vol. 3, no. 8, 2008, pp. 1–205.
12. ATLAS Collaboration, *ATLAS High-Level Trigger, Data-Acquisition and Controls*, tech. report, CERN-LHCC-2003-022, ATLAS-TDR-016, CERN, 2003.
13. F. Bergero and E. Kofman, "A Vectorial DEVS Extension for Large Scale Parallel System Modeling and Simulation," *Simulation*, vol. 90, no. 5, 2014, pp. 522–546.
14. F. Bergero and E. Kofman, "PowerDEVS: A Tool for Hybrid System Real-Time Modeling and Simulation," *Simulation*, vol. 87, nos. 1 and 2, 2011, pp. 113–132.
15. R. Castro, "Integrative Tools for Modeling, Simulation and Control of Data Networks" (in Spanish, extended summary in English), PhD dissertation, Control Dept., Nat'l Univ. Rosario, Argentina, 2010.
16. R. Castro and E. Kofman, "An Integrative Approach for Hybrid Modeling, Simulation and Control of Data Networks Based on the DEVS Formalism," *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*, M.S. Obaidat, Z. Faouzi, and P. Nicopolitidis, eds., Morgan Kaufmann, 2015, chapter 18.
17. T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*, Springer, 2008.
18. X. Chang, "Network with OPNET Simulations," *Proc. 31st Conf. Winter Simulation*, 1999, pp. 307–314.
19. J. Suárez et al., "Computer Networks Performance Modeling and Simulation," *Modeling and Simulation of Computer Networks and Systems: Methodologies and Applications*, M.S. Obaidat et al., eds., Morgan Kaufmann, 2015, ch. 7.
20. T. Colombo et al., "Modeling a Large Data-Acquisition Network in a Simulation Framework," *Proc. Cluster Computing Conf.*, 2015, pp. 809–816.
21. J.L. Burbank, W. Kasch, and J. Ward, *An Introduction to Network Modeling and Simulation for the Practicing Engineer*, John Wiley & Sons, 2011.
22. F. Cellier and E. Kofman, *Continuous System Simulation*, Springer Science & Business Media, 2006.

23. B. Gonçalves and F. Porto, "Managing Scientific Hypotheses as Data with Support for Predictive Analytics," *Computing in Science & Eng.*, vol. 17, no. 5, 2015, pp. 35–43.
24. S. Kulkarni and P. Agrawal, *Analysis of TCP Performance in Data Center Networks*, Springer, 2014.
25. T. Colombo (on behalf of the ATLAS Collaboration), "Data-Flow Performance Optimisation on Unreliable Networks: The ATLAS Data-Acquisition Case," *J. Physics: Conf. Series*, vol. 608, no. 1, 2015, p. 012005.
26. S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Rev.*, vol. 42, no. 5, 2008, pp. 64–74.
27. I. Antcheva et al., "ROOT: A C++ Framework for Petabyte Data Storage, Statistical Analysis and Visualization," *Computer Physics Comm.*, vol. 182, no. 12, 2011, pp. 2499–2512.
28. M.E. Pozo Astigarraga (on behalf of the ATLAS Collaboration), "Evolution of the ATLAS Trigger and Data Acquisition System," *J. Physics: Conf. Series*, vol. 608, no. 1, 2015, p. 012006.

Matías Bonaventura is a MASc in computer science and a PhD student in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, at the Universidad de Buenos Aires, Argentina. Contact him at mbonaventura@dc.uba.ar.


Daniel Foguelman is a MASc in computer science and a PhD student in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, at the Universidad de Buenos Aires, Argentina. Contact him at dfoguelman@dc.uba.ar.

Rodrigo Castro is a PhD in electrical engineering (Universidad Nacional de Rosario, Argentina) and professor in the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, at the Universidad de Buenos Aires, Argentina. Contact him at rcastro@dc.uba.ar.

cn Selected articles and columns from IEEE Computer Society publications are also available for free at <http://ComputingNow.computer.org>.

Keeping YOU at the Center of Technology

myCS




Publications your way,
when you want them.

The future of publication delivery is now. Check out myCS today!

- Mobile-friendly**—Looks great on any device—mobile, tablet, laptop, or desktop
- Customizable**—Whatever your e-reader lets you do, you can do on myCS
- Personal Archive**—Save all your issues and search or retrieve them quickly on your personal myCS site.

Stay relevant with the IEEE Computer Society

More at www.computer.org/myCS

IEEE  computer society