Buffer Provisioning for Large-Scale Data-Acquisition Systems

Alejandro Santos*
Physics Department, CERN
Geneva, Switzerland
Institute of Computer Engineering,
Ruprecht-Karls University of Heidelberg
Heidelberg, Germany
alejandro.santos@cern.ch

Wainer Vandelli
Physics Department, CERN
Geneva, Switzerland
wainer.vandelli@cern.ch

Pedro Javier García
Computing Systems Department,
University of Castilla-La Mancha
Albacete, Spain
PedroJavier.Garcia@uclm.es

Holger Fröning
Institute of Computer Engineering,
Ruprecht-Karls University of Heidelberg
Heidelberg, Germany
holger.froening@ziti.uni-heidelberg.de

ABSTRACT

The data acquisition system of the ATLAS experiment, a major experiment of the Large Hadron Collider (LHC) at CERN, will go through a major upgrade in the next decade. The upgrade is driven by experimental physics requirements, calling for increased data rates on the order of 6 TB/s. By contrast, the data rate of the existing system is 160 GB/s. Among the changes in the upgraded system will be a very large buffer with a projected size on the order of 70 PB. The buffer role will be decoupling of data production from on-line data processing, storing data for periods of up to 24 hours until it can be analyzed by the event processing system.

The larger buffer will allow a new data recording strategy, providing additional margins to handle variable data rates. At the same time it will provide sensible trade-offs between buffering space and on-line processing capabilities. This compromise between two resources will be possible since the data production cycle includes time periods where the experiment will not produce data.

In this paper we analyze the consequences of such trade-offs, and introduce a tool that allows a detailed exploration of different strategies for resource provisioning. It is based on a model of the upgraded data acquisition system, implemented in a simulation framework. From this model it is possible to obtain insight into the dynamics of the running system. Given predefined resource constraints, we provide bounds for the provisioning of buffering space and on-line processing requirements.

*This is the corresponding author

KEYWORDS

CERN, ATLAS, Data Acquisition System, Simulation, Modeling, OMNeT++, Buffering.

ACM Reference Format:
Alejandro Santos, Wainer Vandelli, Pedro Javier García, and Holger Fröning. 2018. Buffer Provisioning for Large-Scale Data-Acquisition Systems. In *Proceedings of The 12th ACM International Conference on Distributed and Event-based Systems (DEBS '18).* ACM, New York, NY, USA, 12 pages.

1 INTRODUCTION

Colliding beam High Energy Particle Physics experiments study physical phenomena by measuring subatomic particles. A complex layout of sensors and special-purpose electronic devices detect the product of particles interaction at the collision point. Analysis of the data produced by these kind of experiments takes time, in the order of several months or even years. Not only because of the vasts amount of data involved, but also because experiment goals are the study of new, never-examined before physical phenomena.

For this reason, data have to be recorded in permanent storage to allow scientists to iterate over the results. However, it is unfeasible to record all the data produced, since the requirements for storage will be enormous. Also, a large portion of the data represents known phenomena. Therefore, data are filtered in real-time before being sent to permanently storage. This initial filtering is done by identifying general aspects of the interesting phenomena for the experiment.

The data-acquisition system (DAQ) is in charge of receiving the data from the sensors, filtering and selecting relevant data, and sending data to permanent storage. The filtering stage is usually called a *trigger* system. The DAQ system implementation often involves a mixture of custom electronics and general purpose computing hardware, like Ethernet networks.

In this and similar contexts, the data rates of the DAQ system exhibit the following behavior during execution:

(1) Data rates are very high, and it is unfeasible to store all produced data.

(2) Data rates vary in time, depending on external conditions that are usually not possible to control.
(3) Data rates may show a cyclic behavior, including a drop to zero for a certain amount of time. This can be due to external or technical reasons.

As a result, many experiments provision their DAQ system for peak data rates. During the time the experiment is either not delivering data or producing data below its peak operation, the processing system will remain totally or partially unused. Future DAQ systems will observe a substantial increase of maximum data rate, resulting in an increase of the gap between the average and peak values for data rates.

For example, the ATLAS experiment at CERN [3], a major experiment at the Large Hadron Collider (LHC), uses different sensors to record collision data. Protons are collided 40 million times per second over periods that can last longer than 24 hours, so that vast amounts of physics data are produced. This data stream has very specific production patterns, and bounds for data rates and objects sizes vary over time. The trigger system of the ATLAS DAQ system [2, 4] is implemented in two stages: while the first stage consists of custom electronics with strict real-time requirements, the second stage is based on a general-purpose multicore computing system, interconnected using an Ethernet network. Under today’s configuration, so-called ATLAS “Phase-0”, the second trigger stage receives data from the sensors electronics on average at 160 GB/s. The future upgrade for the ATLAS experiment, so-called “Phase-2”, will have to deal with an increase in maximum data rates on the order of 30x when compared to the existing system. Such an increase of data rate, even if it will take place in the future, has tremendous implications on the system architecture. However, the LHC is unable to deliver collisions continuously. Stable collisions that contribute for the experiment’s data production will occur around 65% of the time. Figure 1 shows this behavior in the existing ATLAS system. This presents an opportunity to trade-off computing power with storage space.

Previous related work includes [15], which describes a simulation model to study the single-stage buffering system of the existing ATLAS “Phase-0” experiment, a model which in this paper is called *single buffer* simulation model. By contrast, this work is about a *split buffer* simulation model for a large buffer like the one it will be required for ATLAS “Phase-2”, and which is distributed across multiple storage entities.

Two scenarios of operation are studied: when the experiment is delivering data, and when the experiment is not delivering data. In the first case, the system has to both store data in the buffering system, and process incoming data on-line. In the second case, the experiment has to continue the processing of the data leftover in the buffering system. Within each scenario, the size of the data processing system is explored in order to analyze the storage and throughput requirements.

Model validation is done from two sides. First, the *split buffer* model is run and compared against the existing system by using *single buffer* model results [15] as reference, described in Section 4. Second, a small-scale emulated DAQ system was created and the simulation model compared against the output metrics of the small-scale emulated DAQ system, as described in Section 5.

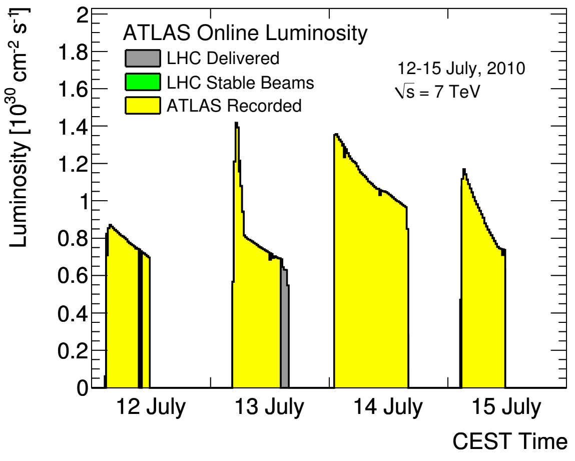

Figure 1: Instantaneous LHC luminosity as observed by ATLAS over a period of several days. For any given physics process, the rate of generated events is directly proportional to the luminosity.

Results for this work are estimates for the usage of the buffering components of the model, and for each scenario of operation. This will provide an estimate for the processing power and networking requirements of the future system. Future provisioning of large-scale DAQ systems can rely on the work proposed here, in order to improve resource provisioning. In general, decoupling data production and data processing can benefit many systems doing intensive data analysis. For example, the new DAQ system of the CMS experiment[5] follows this design, having a large buffer that can hold data for several minutes.

In summary, the contributions of this paper are:

(1) Proposing large-scale storage buffers for data acquisition systems to reduce overprovisioning of the processing system, to allow handling temporary data bursts higher than design rate without data loss, and to improve decoupling of data collection and data processing system.
(2) A simulation model for the DAQ system that is accurate and validated against existing real-world data.
(3) Exploring the operational envelope of a DAQ system, which describes trade-off possibilities in between storage and processing when applying a storage buffer.

The rest of the paper is organized as follows: Section 2 describes the data acquisition process for the ATLAS “Phase-2” system. Section 3 describes the *split buffer* simulation model. Section 4 describes the validation of the *split buffer* simulation model against real data of the existing system, and Section 5 describes the validation of the *split buffer* simulation model against the small-scale emulated DAQ system. Section 6 describes the operational envelope to characterize resource utilization of the system. Section 7 provides more details regarding related work on DAQ system and storage systems. Finally, Section 8 provides the conclusions of this paper.

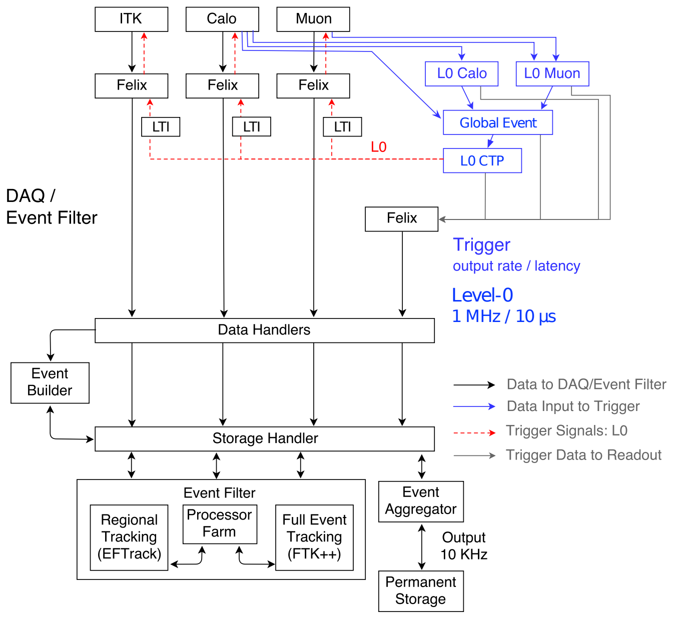

Figure 2: ATLAS Phase-2 Upgrade dataflow architecture. Buffering of incoming data to be processed is split into the Data Handler and the Storage Handler systems.

2 BACKGROUND

This section mainly describes the foreseen ATLAS “Phase-2” data acquisition system, but also overviews the DAQ of other systems.. Details for the ATLAS “Phase-0” system, the corresponding simulation model and results can be found in [15]. Figure 2 shows the dataflow organization of the ATLAS “Phase-2” DAQ.

2.1 The Interfill Period

The accelerator, for both technical and physics reasons, has to stop delivering collisions after a finite period of operation, and resuming production requires several hours of preparation. The time period in the LHC cycle when it is not producing data is called the *interfill period*, which presents an unavoidable efficiency limitation. The *duty cycle* or *cycle efficiency* is the time percentage when collisions are delivered. Real data for the efficiency of the existing system can be seen in Figure 1, where the actual pattern for the cycle does not follow a stable pattern. A mixture of both long and short cycles are present, and within each cycle there is a lot of variation affected by many external factors.

This interfill period presents an opportunity to trade-off processing power with storage space. The processing system can be downscaled below the required size to process all incoming data in real time. Below this point, the system is required to save data in the buffering system to avoid data losses. Even tough the instantaneous rates for production and processing will not match, the buffering system is used to balance the difference over the time period of the full cycle. The processing system has to be large enough to allow the completion of the processing of the data during this period, and before the next data production cycle starts. This strategy can be implemented in general in any DAQ system, and is a strong choice for ATLAS “Phase-2”. In the existing ATLAS “Phase-0” system, the buffering system has enough space to accommodate for fluctuations in the incoming data rates and can buffer incoming data in the order of seconds. It is implemented as a single RAM-based storage space.

2.2 ATLAS Upgrade Design

The ATLAS DAQ will receive data from many *sensors*. Different sensor families provide different data, varying in formatting and size. The data from all sensors corresponding to a single particle bunch crossing is called *event*. An ATLAS event is composed by many *fragments*. Fragments are defined as logically grouped ATLAS detector elements optimized for efficient usage of DAQ resources.

In ATLAS “Phase-2”, it is anticipated the DAQ will select and process 1 million events per second. Approximately only 10 thousand events per second will be retained for offline analysis. Thus, ~99% of the data will be discarded by the DAQ system.

A dedicated custom-hardware electronics system, known as Level-0, will perform the initial reduction from the collision rate of 40 MHz to 1 MHz. Upon selection by the Level-0, data from the detector sensors are pushed through approximately 17000 high-speed serial links into FELIX [16]. FELIX acts as interface between these links and a general purpose network. In the baseline design it will be implemented with several hundred servers equipped with custom PCIe interface cards.

From FELIX the event data is transferred to the Data Handlers, a set of commodity servers where sensor-specific post-processing takes place. The Data Handlers are followed by a large storage system, the Storage Handler, decoupling the data movement and aggregation from the filtering. The latter takes place in a distributed computing farm called the Event Filter. At the anticipated input rate of 1 MHz, the Data Handler will receive a data rate of about 6 TB/s.

For each event, the Event Filter will incrementally fetch and analyze data fragments until a decision on whether to accept or reject the event is taken. Accepted events will be organized in files and transferred to off-site permanent storage. Data of rejected events will be deleted from the system.

The processing time of an event will follow a very complex distribution, composed by many populations, as driven by the difference physics processes taking place in the ATLAS detector.

A meta-data system, implemented on top of Data Handler and Storage Handler will keep track of events and fragments. It will also implement the Event Builder functionality, which involve logically or physically joining all fragments corresponding to one event. Finally the meta-data system will manage the assignment of events to the Event Filter.

In this paper, a buffering model distributed across Data Handler and Storage Handler is analysed. In this scheme short-term buffering takes place in the Data Handler, while the Storage Handler provide long-term buffer space. The meta-data system in this case provides transparent access to the data for the Event Filter, while managing the transition between short-term and long-term storage.

Indeed data fragments corresponding to events experiencing a long processing time in the Event Filter have to be moved from the Data Handler to the Storage Handler to create space for new incoming data.

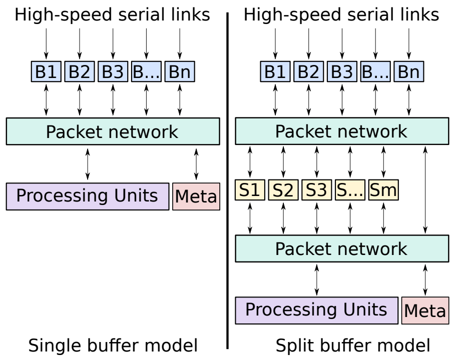

Figure 3: Comparison between *single buffer* and *split buffer* models. In both models each buffer B_i receives fragment data synchronously, while in the split model each buffer S_k receives fragment data asynchronously.

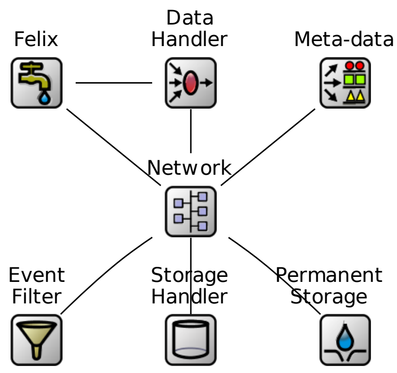

Figure 4: OMNeT++ model for the *split buffer* simulation.

2.3 Other DAQ systems

Another example of a DAQ is at the ANTARES neutrino telescope [1]. As observed in [1], "*The data acquisition system of the [ANTARES] detector takes care of the digitisation of the photo-multiplier tube signals, data transport, data filtering, and data storage.*" The ANTARES detector consist of 900 sensors located 2.5 km undersea. The data output of the detector is ~0.5 GB/s. An initial filtering selects physical signals from the background phenomena, and this processing is done in a farm of standard PCs.

3 SIMULATION MODEL CONSTRUCTION

This section introduces the *split buffer* model, followed by a detailed comparison against the *single buffer* model and a description of the used simulation environment.

A high-level overview between the *single buffer* and *split buffer* models is shown in Figure 3. In both cases, data arrives synchronously to the buffering machines B_i via high speed serial links. Each serial link transports the stream of data for one kind of fragment. Fragments are buffered in the machine they arrive. For the analysis of each event, processing units request fragments to the buffering machines where they are located. Coordination is done by the meta-data system *Meta*.

Parameter	Description
Level-0 rate	Event output rate for the Level-0 trigger
Total cores count	Total number of CPU cores in the Event Filter
Fast-reject delay	Time for the fast rejection of an event
Full-reject delay	Time for the full processing of an event
Fast-reject probability	Probability for an events for being fast-rejected
Full-reject probability	Probability for an events for being full-rejected
Data Handler count	Number of Data Handlers
Request count (fast)	Number of Data Handlers requested per event for fast-rejected events
Request count (full)	Number of Data Handlers requested per event for full-rejected events
Fragment Size	Average size of fragment per event
Network overhead	Network overhead factor
Max. frags. in Data Handler	Maximum number of fragments stored in Data Handler
Cycle time	Total length of the LHC cycle
Cycle duty	Length of the LHC production cycle

Table 1: Input parameters description for the *split buffer* simulation.

In the *single buffer* model, there is only one stage of buffering. Data arrives synchronously to the buffering machines B_i and the processing units request data directly to this buffer stage. Fragments are stored in buffering machines B_i for periods of time in the order of seconds, for the processing system to analyze the event data. Fragments from the buffering are removed shortly after the event is processed. There is no synchronization required to be performed between buffer machines and processors, because fragment data arrives to all the machines at the same time. Processing units also know statically the location of each fragment, because is implied by the physical serial link connections. Coordination is done by the *Meta* system by tracking individual events, and assigning the events to processing units.

In the *split buffer* model, there are two buffering stages: the intermediate buffer machines B_i, and the large buffer machines S_k. Fragments are stored in the intermediate buffer space for short term periods, in the order of seconds. For the large buffer, data is stored for periods of many hours. This large buffer enables the possibility to have a processing system with a size smaller than the required to process data at the peak rate. Data arrives synchronously via serial links to the intermediate buffering machines B_i. From the intermediate buffers, data is asynchronously moved to the large buffering machines S_k through a packet switching network. There is no implied or static mapping between the machines from the first stage of buffering, B_i, to the second stage of buffering, S_k. And processing units process events by reading fragments from the large buffering machines S_k. Thus, the location of the fragments needs to be tracked and synchronized. Being an asynchronous operation, fragments can be located in the intermediate buffers, in the large buffer, or in both. Therefore, both processing units and machines from the first stage of buffering have to communicate with the meta-data system, in order to synchronize the location of the data for writing and reading fragments.

Figure 4 depicts a high-level overview of the *split buffer* simulation model. *Felix* is the high-speed serial link system. The first layer of buffering is depicted as *Data Handlers*, and the second layer of buffer is *Storage Handler*. Table 1 shows simulation input parameters. The simulation model follows the description of the ATLAS data acquisition system, with some simplifications as follows.

- The network is assumed to be ideal, with no packet drops. The *network overhead* is an input parameter describing in percentage how much more data is used for the protocol layers, i.e., Ethernet, IP, and TCP overheads. This simplification is possible to be followed since in the existing “Phase-0” system a flow control mechanism is used, based on the credit system described in [9]. It mitigates the TCP incast problem by minimizing the number of TCP retransmissions. It is expected that some form of flow-control mechanism continues being part of the system in future upgrades.
- Data delivery in the network is also assumed to be infinite in bandwidth, since the network has to be able to handle the throughput of data as sent by the detector electronics. Also, the network latency is set to zero.

The model presented in Figure 4 shows the model’s modules and relationships for the flow of messages. For all modules except the Data Handler module, there is one instance. The Data Handler module’s count is configurable as a simulation input parameter. At start-up, the Event Filter module informs the Meta-data module of the number of available CPU cores. The Network module simply forwards messages from a source to a destination module, very much like a network switch.

A typical data production looks as follows:

(1) Data production begins at the Level-0 trigger, which sends fragments through the *Felix* serial link to all Data Handlers.
(2) Data Handlers, then, inform the Meta-data module about the reception of a new fragment.
(3) When a CPU core in the Event Filter is available and all the fragments of an event are stored in one or possible multiple buffers, the Meta-data module assigns the event to the Event Filter.
(4) Then, the Event Filter requests the location of fragments from the Meta-data module, and upon reply the fragments are requested from the corresponding buffers. Once all fragments are received, processing of the event is simulated by a random delay drawn from an configurable distribution: exponential or normal.
(5) In the simulation, events are divided in two populations: either “fast” events or “full” events. For each event assigned to the Event filter, it is marked as one of the two kinds with a configurable probability. The two kinds differ in the delay time to simulate the processing of the event, and the number of fragments associated with an event. Both values are configurable (see Table 1).
(6) Fragments are moved from the Data Handler to the Storage Handler when the number of fragments in the Data Handler is above the configurable threshold (see Table 1). The movement operation is initiated by the Meta-data module but the data is sent by the Data Handler to the Storage Handler.
(7) Once an event is processed, it is either accepted or rejected. Fragments of accepted events are sent to the Permanent Storage. Then, independent of being accepted or rejected, the event’s fragments are removed from the buffers, either by the Data Handlers or the Storage Handler.

3.1 Setting Parameters for Single Buffer and Split Buffer Simulation Models

Compared to the *split buffer*, the *single buffer* simulation model uses histograms to produce specific probability distributions. They are used to determine: data request patterns, processing time, and the number of fragments associated with an event.

Three sets of histograms describe:

(1) The rate of requests per second for each buffer machine
(2) The number of data fragments requested for each buffer machine
(3) The average time it takes to process an event

The first two histograms are used to select: the random buffer machine to make a request of data, the number of requests made for each incoming event, and the number of fragments requested for each incoming event. In the *split buffer* model, an analytical model is used combining the total event size in bytes and the first two histograms as a probability distribution, and to obtain for each event the average size in bytes of data sent through the network.

The third histogram used as input parameter for the *single buffer* model can be greatly simplified by using the following formula (see also [15]). According to this formula, the average processing time over N_p records for a processing time p_i and the number of processed events q_i is:

$$\frac{\sum_{i=1}^{N_p} p_i q_i}{\sum_{i=1}^{N_p} q_i} \tag{1}$$

On average, over the selected time interval for the data to validate the *single buffer* model, the average processing time is $\sim$ 180 ms.

In the *split buffer* simulation model the processing time is modeled as a single exponential distribution with this value as average, whereas in the *single buffer* model there are ~40 k different exponential distributions.

In the *single buffer* model, the output bandwidth is modeled with an overhead increase of 46 bytes for every 1454 bytes for each response from the buffering system to the processing system, since the *maximum transmission unit* of Ethernet networks is 1500 bytes. The value of 46 bytes is the overhead size of the protocol headers sent through the network. In the *split buffer* model, network overhead is calculated using a factor of $1500/1454 \approx 1.031$, shown in Table 1 as a simulation input parameter.

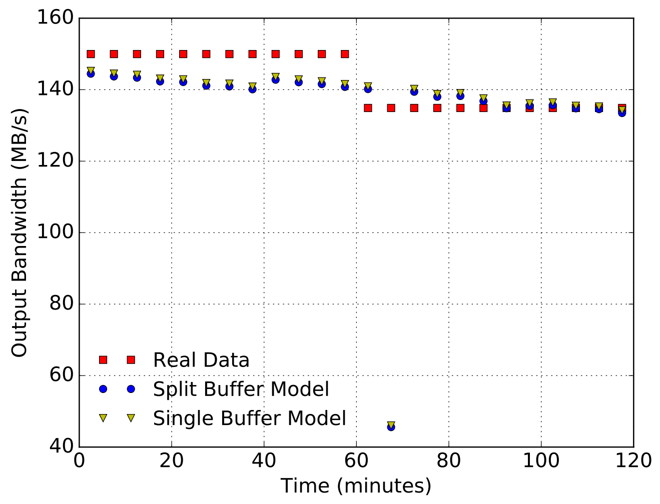

Figure 5: Comparison of operational data and simulation model results for the output bandwidth of the buffering system.

Metric	Simulation model Error Single buffer	Split buffer
Output bandwidth	4.3%	4.8%
Number of fragments	5.4%	4.2%
Active Processing Units	1.3%	< 0.1%

Table 2: Simulation error based on a comparison of operational data and results from simulation for both models.

3.2 Model implementation

Both models are built using OMNeT++ [17, 21], a robust and user-friendly discrete-event simulation framework. OMNeT++ usage is prominent in the network simulation community. Simulations are defined by C++ modules, and relationships between modules are defined by configuration files.

The previously described parameter simplifications also result in an easier to understand and easier to use model. Furthermore, configuration files of the simulation are smaller by several orders of magnitude, and there is a big improvement of the simulation performance. In the *split buffer* model, one simulated second running at the data rates of the existing system requires about 15 real seconds of wall-clock time. For the *single buffer* model, one simulated second requires about one hour of wall-clock time. A large contribution of performance improvement comes from the simplifications in the modeling. Instead of using histograms as source of randomness the built-in exponential random number generator of OMNeT++ is used, which is a faster way to produce random numbers.

4 SIMULATION MODEL VALIDATION USING OPERATIONAL DATA

To validate the *single buffer* model, there exists a huge amount of operational data from the current ATLAS "Phase-0" experiment.

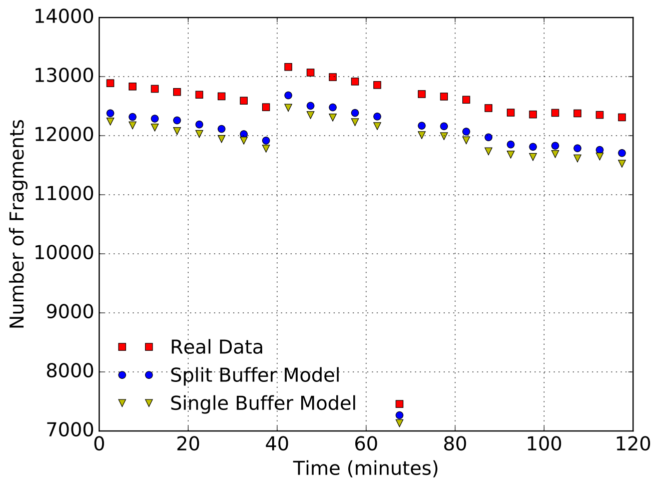

Figure 6: Comparison of operational data and simulation model results for the number of fragments stored in the buffering system.

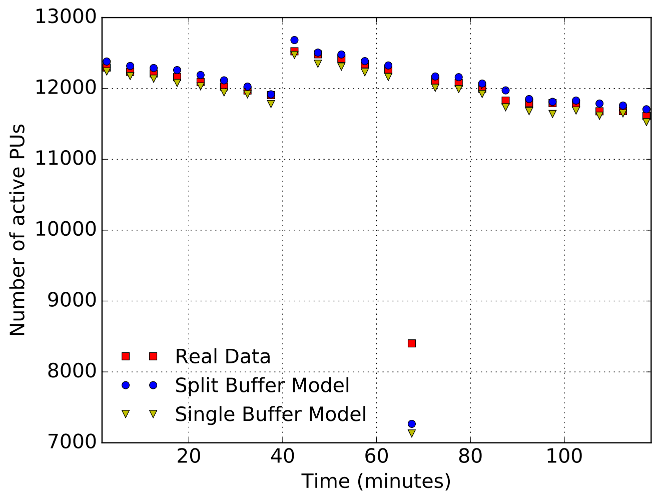

Figure 7: Comparison of operational data and simulation model results for the number of active processing units.

Because there is obviously no operational data available to validate the *split buffer* model, we separate this validation into two parts. First, and part of this section, the *split buffer* model is validated against the same ATLAS operational data as the *single buffer* model, by running the *split buffer* model using *single buffer* settings. In the next section, we will extend this validation by covering the case of multiple storage buffers.

Thus, we configure for first buffering stage of the *split buffer* we set the number of Data Handlers to one, mimicking *single buffer* configuration. We follow the methodology described in [15] for the *single buffer* model, but extend the validation for the *split buffer* model as described in Section 3. The simulation is run for 60 simulated seconds. The values compared for the validation are averages, where operational data values correspond to the average of the five minutes, while simulation results corresponds to the average of the 60 seconds of simulation.

The output of both simulations is compared against operational data derived from the existing ATLAS "Phase-0" system, in particular related to the main metrics of the system:

- Average output bandwidth metrics of the buffering system in MB/s
- Average number of data fragments stored in the buffering system
- Average number of active processing units

Data and results corresponds to an interval of 2 hours, divided in 24 intervals of 5 minutes each. Results are shown in Figure 5, Figure 6, and Figure 7, respectively. The three figures show an outlier at minute ∼ 70, where the real system had to stop accepting data due to external conditions.

Figure 5 shows the comparison between simulation and real metrics for the average output bandwidth of the buffering system. Real data is archived with a limited resolution of one hour, which explains the shape of the data as an step function of one hour. For this reason, the simulation does not match real data point at minute ∼ 70.

Figure 6 shows the comparison between simulation and real metrics for the average number of fragments stored in each buffer. There is a systematic bias in the results between the real data and simulation results, because in both models several latencies present in the real system are set to zero. They include network and software latencies.

Figure 7 shows the comparison between simulation and real metrics for the average number of active Processing Units. This number directly depends on the processing time of the events in the system and the input data rate.

Generally, we observe a very good match in between operational data and simulation results for both models, resulting in errors of 5.4% or less. Table 2 summarizes the results from this validation.

5 VALIDATION OF THE STORAGE BUFFER

This section describes the validation against the small-scale emulated DAQ from Section 5.1. The previous section validated the *split buffer* model for a setting similar to the *single buffer* model. However, this setting basically neglects the presence of a distributed storage buffer. As there is no operational data available to utilize such a distributed storage buffer, in this section we first introduce an tool that allows us to generate such data using a small distributed computing system. Then, we use these results for a validation against the simulation results for the *split buffer* model.

5.1 Small-scale Data-Acquisition Emulator

To obtain operational data for the validation of the split buffering concept, we develop a software tool that emulates a data acquisition system as described in Section 2. It is implemented in the Python programming language, and designed as a distributed system. All communication is done using the ZeroMQ [13] message passing library. There is no flow control mechanism for the data sent through the network, as it is expected that the network can guarantee drop-less packet forwarding by appropriate provisioning and a credit-based flow control as described in [9].

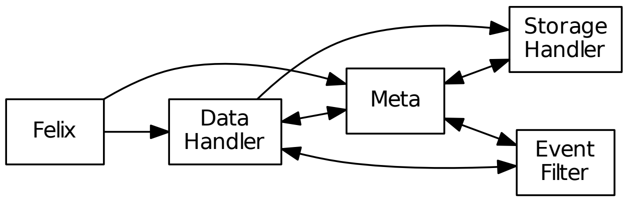

Figure 8: Architecture for the small-scale emulated DAQ implementation. Each box represents an application, and arrows represent the flow of messages. Some applications may be instanced more than once.

Figure 8 shows the software architecture, based on multiple applications, and the flow of messages. There is one instance of each of the Felix, Meta, and Storage Handler application. The number of instances for the Data Handler and Event Filter applications is configurable. The different applications behave as follows:

- The Felix application represents the source of data fragments and handles their creation at a configurable rate. Data arrives to the system in the form of events, where an event consists of multiple fragments, one for each Data Handler.
- The Meta application handles the meta data of the system. It tracks available fragments in both Data Handler and Storage Handler, and the available Event Filter applications. It erases fragments from the Data Handler and Storage Handler, and assigns events to the Event Filter. Events are assigned from a list of unassigned events, giving priority to the ones buffered in the Data Handler. Only when there are no events in the Data Handler, events buffered in the Storage Handler as selected.
- The Data Handler application is the first buffering stage. It notifies the availability of fragments to the Meta application, answers fragment requests from the Event Filter, and sends fragments to the Storage Handler.
- The Storage Handler application is the second buffering stage. It receives fragments from the Data Handler, and answers fragment requests from the Event Filter. Fragment movement from the Data Handler to the Storage Handler is initiated by the Meta application, although fragments are sent directly from one to another. Only fragments of unassigned events are moved.
- The Event Filter application receives work, issues location requests to the Meta application, and requests fragments to the Data Handler or Storage Handler. Each Event Filter processes one event at a given time, and the processing of the fragments is emulated by a time delay.

5.2 Single Execution of the Small-Scale DAQ System

The validation of the Data Handlers is based on metrics including event rate, number of events stored in Data Handlers and Storage Handler, and input and output bandwidth for the Storage Handler.

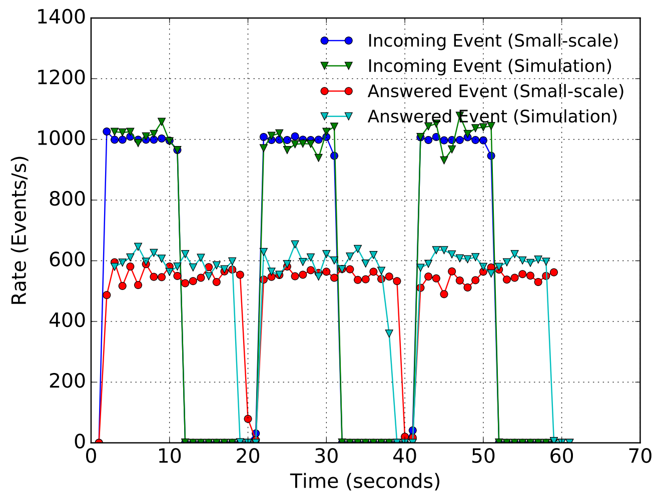

Figure 9: Comparison of event rates, for a single execution of the small-scale DAQ and the split buffer simulation model. The values are sampled once a second for 60 seconds.

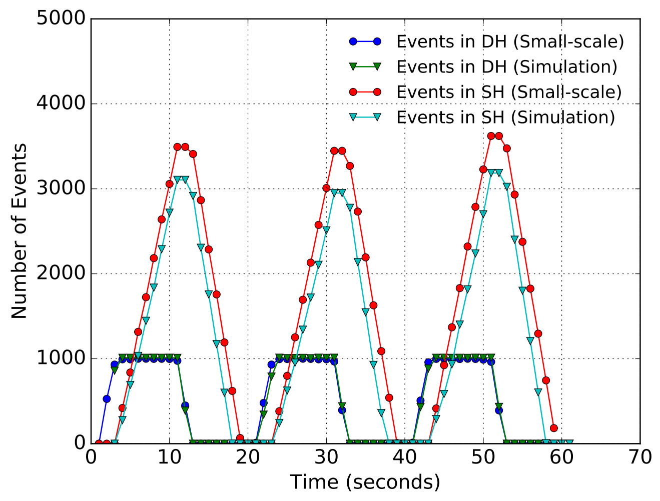

Figure 10: Comparison of number of events stored in the Data Handlers (DH) and the Storage Handler (SH) metric, for a single execution of the small-scale DAQ and the split buffer simulation model. The values are sampled once a second for 60 seconds.

The small-scale data-acquisition emulator is executed on a dedicated Gigabit Ethernet network of five GNU/Linux computing nodes, each having dual Intel Xeon CPU E5540 with 16 cores. It is ensured that the system is not loaded otherwise. Time is measured using the `time.time()` Python function, which on Linux has a resolution of 0.001 milliseconds. We set the number of Data Handlers to 10, and the number of Event Filters to 12. There is one instance each for the Felix, Meta, and Storage Handler.

The experiment is running for a period of 60 seconds, with an incoming rate of 1,000 events/s. Being a small-scale experiment and

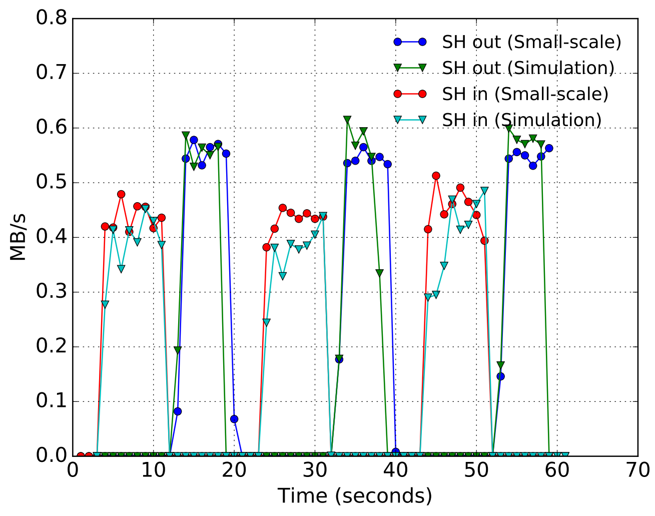

Figure 11: Comparison of Storage Handler (SH) input and output bandwidth, for a single execution of the small-scale DAQ and the split buffer simulation model. The values are sampled once a second for 60 seconds.

not a complete system, it represents a scaled-down version of the full ATLAS DAQ system, with incoming data rate set to a fraction of the "Phase-2" system. The cycle begins with a period of 10 seconds of data production followed by a period of 10 seconds where no new events arrive to the system. This pattern repeats three times to complete 60 seconds. Processing time for events is 20 ms, and the maximum number of events stored in the Data Handler is 1,000. Thus unprocessed fragments that remain in the Data Handler for more than one second will be sent to the Storage Handler.

We replicate this configuration for the *split buffer* model. Figure 9, Figure 10, and Figure 11 show the results of a single execution for both the small-scale system and the *split buffer* simulation.

Figure 9 shows the rate of both incoming and acceptance of events, sampled at the Meta application. The rate of incoming is the rate at which the Felix produces events, and the acceptance is the amount of events processed by the Event Filters. Figure 10 distinguishes the related number of events stored in both types of buffers, the Data Handlers (DH) and the Storage Handler (SH). Finally, Figure 11 reports the related input and output bandwidth for the Storage Handler buffering stage.

Table 3 shows latency results for the data-acquisition emulator software. Values are measured at the Meta application, averaged for all events. The total time represents the time since an event is sent to the Event Filter until the event answer is received. The processing time is the time it took to process an event. Since the software is an emulator, no real processing occurs and the time is a delay call to the operating system. Then, the overhead is the total time minus the processing time.

Data Handler time and Storage Handler time values from Table 3 correspond to the average time an event remains in the system until is permanently deleted. The first value, Data Handler time, corresponds to events which remain entirely in the Data Handler system.

Name	Latency average value
Total time	21.7 ms
Processing time	20.0 ms
Overhead time	1.7 ms
Data Handler time	0.99 s
Storage Handler time	9.98 s

Table 3: Latency results for the small-scale emulated DAQ experiment.

Contrary, the second value, Storage Handler time, corresponds to events which are in between stored in the Storage Handler.

5.3 Non-zero Overhead Latency

Results from Table 3 shows an overhead latency of 1.7 ms, and Figure 9 shows a discrepancy between simulation and Small-scale emulator results. This difference can be attributed to the overhead latency in the simulation set to zero. Running a new simulation that includes a calibration latency for the overhead of 1.7 ms thus having a total time of 21.7 ms instead of 20 ms does produce results which agree between simulation and the small-scale emulator. Figure 12 shows simulation results in this new scenario. Comparing Figure 9 and Figure 12, the peak number of events goes from $\sim$ 3000 to $\sim$ 3600.

Determining the overhead latency can be a challenge. Already having an system which is built and running allows to simply measure the overheads of the system. For example, this is the case of the current small-scale emulator experiment. However, in general, if the system is not yet built and the specific choices for hardware and software infrastructure are unknown, some other method has to be used to determine the overhead. One option is to build a small-scale system which follows the technological elements of a full-scale system, and to use this system to measure and extrapolate overheads. A second option is to use the simulation model to either model and analyze concrete components of a future system, or scan for a range of overheads within some tolerance range in order to understand the implications of the changes in overheads.

6 OPERATIONAL ENVELOPE

In the previous section, the *split buffer* simulation model is validated against real operational data. In this section, the *split buffer* simulation model is used to reproduce and expand over the original results, by studying the behavior of the system under scenarios of increasing complexity. It is first studied with the data production at constant rate, then with data production as a cycle, and finally with incremental variance on the processing time. This last experiment is of particular interest since in the future ATLAS DAQ system the distribution of the processing time for the events will have a large variance.

Resource utilization of the system is explored by having the data rates of the upgraded ATLAS system, at 1 million events per second. The simulations were executed in four dual Xeon E5540 computers, using 12 cores in each computer and taking about 10 hours in each computer to complete. For each simulated second, the simulation model requires about 90 seconds of wall-clock time.

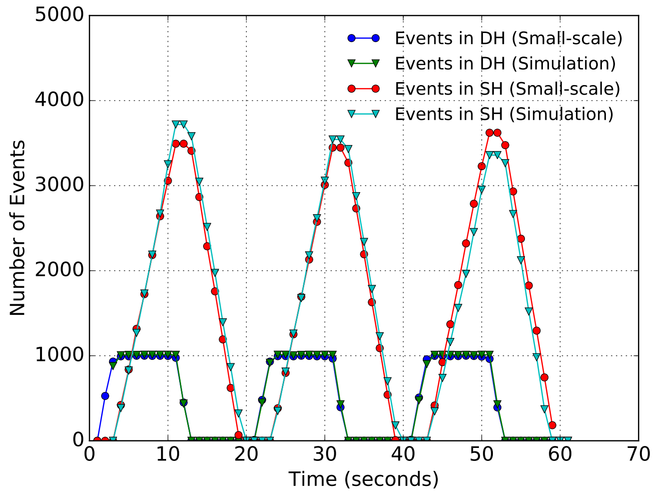

Figure 12: Comparison of number of events stored in the Data Handlers (DH) and the Storage Handler (SH) metric, for a single execution of the small-scale DAQ and the split buffer simulation model. The values are sampled once a second for 60 seconds. Average total processing time in the simulation is 21.7 ms, by including a calibration value for the overhead latency.

The operational envelope describes the overall system utilization as a function of distinct resources. In this case, two resources are considered: compute power and buffering space. When both resources are fully utilized, the system is at the peak utilization efficiency.

Thus, the efficiency f_e is defined as the product of memory usage f_m, compute usage f_p, and fraction of discarded events f_d. The first two factors are the ratio of used resources over total resource amount, while the discarded event factor is the ratio of events discarded for the lack of storage space.

$$f_e = f_m \times f_p \times (1 - f_d) \tag{2}$$

It is important to notice that in high-energy physics experiments it is often fundamental to minimize f_d, at the cost of over-provisioning other resources. At the same time, from a system design point, leaving f_d unconstrained allow understanding its dependence from the system parameters.

In the following, the utilization efficiency will be investigated as a function of the deployed buffer size. It is anticipated the efficiency will reach a peak value at a buffer size defined by various operational parameters. In fact, on the left side of this peak the system does not have enough buffer space to store new events, and existing events are not removed fast enough. Thus, data loss due to event discarding will occur, and compute usage will likely be below maximum. On the right of the peak, compute resource usage will remain constant, but the buffer occupancy will decrease. If enough compute resources are available the system will process all incoming data in real time, with no data loss. The operational envelope value reaches 100% efficiency only when processing capacity and maximum buffer

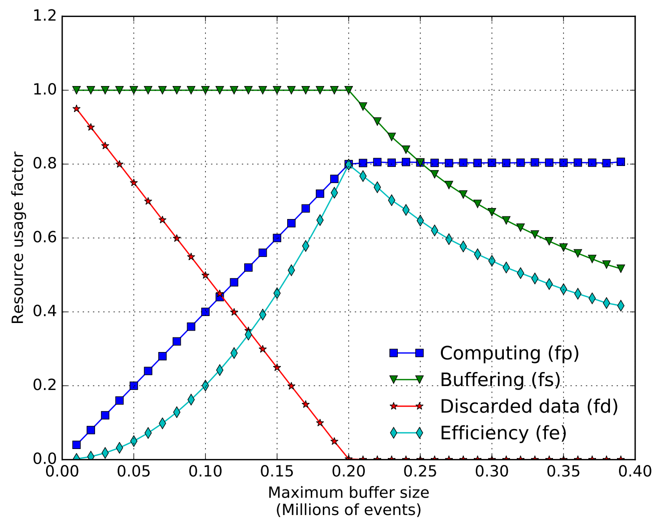

Figure 13: Estimation of the operational envelope as a function of available buffer size. The model is configured with 250,000 CPU cores, a processing latency of 200 ms, and fixed input rate of 1,000,000 events/s. For each data point the simulation was configured with an available buffer space as reported on the horizontal axis. Each simulation is run for 60 simulated seconds.

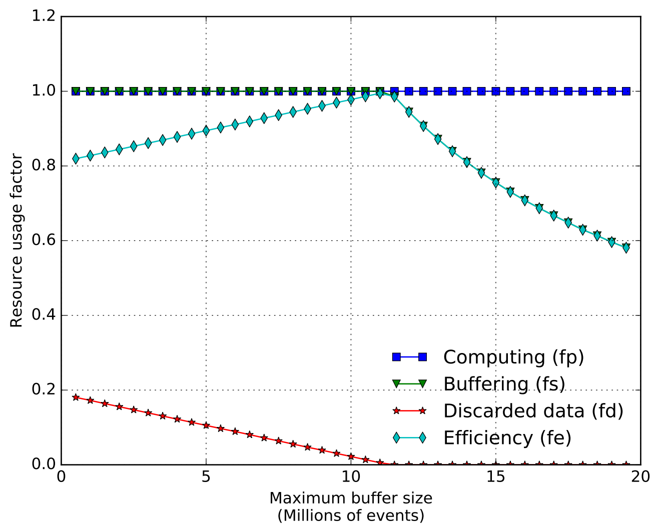

Figure 14: Estimation of the operational envelope as a function of available buffer size. The model was configured with 150,000 CPU cores, a processing latency of 200 ms, and input rate of 1,000,000 events/s at a 75% duty cycle. For each data point the simulation was configured with an available buffer space as reported on the horizontal axis.

space are at the minimum, based on a processing without discarded events.

The parameters affecting the operational envelope are the input event rate, the event size, and the overall system latency. The latter directly depends on the event processing time. In this context, the buffer space is described in terms of the *number of events in storage*, which represents the average space required to store one event.

One can consider the operational envelope from an analytical perspective, as it was done in [15]. On the other hand the use of a simulation model offers the possibility to explore other metrics and constraints. In particular, by limiting the amount of available buffer space in the simulation model, the usage of compute and buffer resources can be explored. In addition the *data loss*, which is an important metric for a DAQ system, can be analyzed as well as the impact of different distributions for the input parameters.

6.1 Data Production at Constant Rate

Figure 13 shows the results of the *split buffer* model with a sweep over different values of the available buffer space. Each simulation is configured with a number of 250,000 CPU cores, and an input event rate of 1,000,000 events/s. The processing latency follows an exponential distribution with an average value of 200 ms. In total 40 simulations were run, starting from a maximum of 1 event in storage and ending at a maximum of 400,000 events in storage. Each simulation is run for 60 simulated seconds with a cycle of 100% production rate.

The system reaches the peak utilization efficiency of the operational envelope when enough buffer space is available to serve all incoming events in real time. The peak efficiency is not 100% since the compute capacity is over-provisioned. As expected, after the peak, the efficiency deteriorates due to the buffer space being over-provisioned. Analytically the amount of buffer space required to reach the operational envelope peak is 20,000 events. The simulation results on the other hand show a slightly higher value, because at that point the data loss is small but non-zero, of about ∼0.5%. The cause of this difference is the variance of the exponential distribution used for the processing time.

6.2 Data Production as a Cycle

This experiment introduces a cycle in the simulation, following what's been discussed in Section 1. The possibility to make a trade-off between storage and compute power depends on the availability on a data production cycle. Having a duty cycle below 100% means that it behaves like a square wave. In this experiment, the duty cycle in the production of data is 75%. During the 60 simulated seconds, data arrives for 45 seconds, followed by 15 seconds where no events arrive to the system.

Figure 14 shows the results for a system operating with non-constant input rate. In total, 40 simulations were run, starting from a maximum of 1 event in storage and ending at a maximum of 20,000,000 events in storage. Each simulation is configured with 150,000 CPU cores, processing latency of 200 ms and input event rate of 1,000,000 events/s with a duty cycle of 75%. The processing latency follows an exponential distribution.

The system was purportedly configured with compute power insufficient for real-time operation. Due to the 75% duty cycle, the buffering system absorbs the difference between the instantaneous incoming rate and the instantaneous processing rate. Therefore,

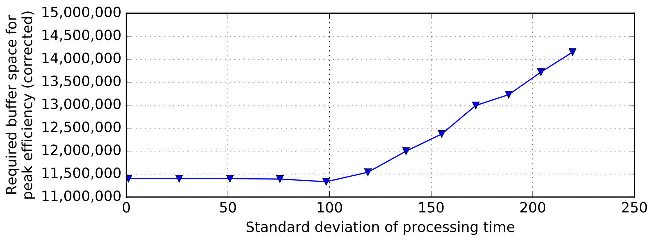

Figure 15: Required buffer space for peak efficiency as a function of the processing time distribution width. The model is configured with 150,000 CPU cores, processing latency of 200 ms following a normal distribution, and input rate of 1,000,000 events/s. Each data point is the buffer space corresponding to the peak utilization efficiency. The measured buffer spare is normalized to the measured average processing time.

peak efficiency is expected when enough buffer space is available to absorb all incoming data with no data loss.

Since the incoming data can not be all processed in real time, the amount of space required to buffer incoming data directly depends on the amount of time the system was producing data.

6.3 Data Processing Variance

This experiment explores a large variance in the processing time of events. In the real ATLAS system, the processing time of an event follows a very complex distribution due to the physical nature of the experiment being studied. This distribution has a large variance. Many events take a short amount of time to analyze, in the order of hundreds of milliseconds, but some non-negligible set of events may take several minutes to analyze.

Figure 15 shows the required buffer space as a function of the processing time distribution width. For the this study the processing time follow a normal distribution whose average value is 200 ms. In total 347 simulations were executed for 60 simulated seconds, starting from a maximum of 10,000,000 events in storage and ending at a maximum of 20,000,000 events in storage. The processing time distribution variance varies from 1 ms to 300 ms. Each simulation is configured with 15,000 CPU cores and an input event rate of 1,000,000 events/s with a duty cycle of 75%.

For each simulation the buffer space required to achieve peak utilization efficiency is evaluated. The result are then corrected by measured effective processing time. In fact, negative processing time values, as generated by large variance settings, are nonphysical. Neglecting these, results in a variable effective average processing time. In order to compare the results of the different simulations, the measured required buffer space needs to be normalized by the processing time variation.

A large variance has a meaningful impact on the storage requirements, as can be seen in Figure 15. Increasing the variance produces in an increase of the storage requirements. The use of a simulation model enables the practical study of the behavior of the system under this large variance of processing time conditions.

7 RELATED WORK

Data acquisition systems. A detailed description of data acquisition systems can be found for example in [19] for the ATLAS experiment, in [5] for the CMS experiment, in [11] for the LHCb experiment, and in [7] for the ALICE experiment, all four experiments located at CERN. The data acquisition system of the ANTARES experiment is described in [1]. In these works, the description of data acquisition systems is provided with much more detail when compared to the one in Section 2.

ATLAS simulations. Related work regarding simulations of the ATLAS data acquisition system can be found in [8, 9] and [6], where some aspects of the ATLAS network were studied. Specifically, studies about network latencies, TCP retransmissions, and the TCP incast pathology can be found. A more general study of the existing ATLAS "Phase-0" buffering system can be found in [15]. They are validated with real ATLAS "Phase-0" data.

Other ATLAS simulation studies are available in [10, 18], which were made before the ATLAS data acquisition system was implemented. In such studies, the focus was rather to understand the consequences of some of the design choices that were made, and in particular how these choices affect system latency. Contrary, in the present work a general overview of the system is the focus of study.

Storage systems. In this paper, the buffer space study does not involve concrete technological solutions. However, future directions of this paper will consider more specific storage technologies and how this impacts system behavior.

Distributed file systems can be scaled to provide large amounts of storage and throughput. Research about performance analysis and characterization for large storage systems implemented with a distributed file system can be found for instance in [20]. The authors provide a general model to predict the performance of a storage system, and offer the Hadoop distributed file system as an example. A concrete example of an existing storage system is the Trinity supercomputer [12], which utilizes a distributed file system for its operation. It has an usable storage of 78 PB and 1.6 TB/s of bandwidth. Alternative data storage systems like Apache Cassandra [14] provide a service to store large amounts of structured data. The service can be scaled by increasing the number of machines with no single point of failure.

8 CONCLUSIONS

A new simulation model is presented, which is validated with both existing operational data from the ATLAS experiment of the LHC and with a small-scale version of a data-acquisition software emulator. Validation results agree between simulation and their counterparts in the real and emulated system. The simulation model is accurate and the performance is improved compared to other models.

The trade-off between storage and compute power requires the availability of a data taking cycle, in order to allow the processing system to finish the analysis of the data in the buffers. Bursts in the incoming data rate can be handled by trading off compute power with buffer space.

The buffering system introduces an abstraction layer between data production and data processing. This approach has potential benefits for the maintenance and evolution of the overall system. The underlying technologies of the buffer can change with a minimal impact on the design of the rest of the components.

The operational envelope of the system characterizes a high-level overview of resource utilization of the system. Three scenarios are studied: constant rate of both production and processing, variable rate of production, and rate of processing with a large variance with a non-exponential probability distribution. For the simple cases, results can be predicted with analytical methods. However, a simulation model enables the study of a system with complex interactions between components and complex behaviors. For example, bursts in the incoming data rate can be more practically studied with a simulation model, as well as complex probability distributions for the processing time of data.

One challenge to the accurate simulation of data-acquisition systems corresponds to the prediction of the overhead latency of the system. If the system is already built, the overhead latency can be measured in a similar way it was done for Table 3. However, if the specifics of the system like the hardware and software infrastructure are yet unknown, one option is to build a small-scale system which mimics the final version of the system. This approach will allow to measure the overheads introduced in the system and the fine-tuning of the simulation of the full-scale system. Another option is to use the simulation model to scan for a range of overhead values in order to include tolerance margins in the sizing of the system design.

ACKNOWLEDGMENTS

This work has been partly supported by the Spanish MINECO and European Commission (FEDER funds) under the project TIN2015-66972-C5-2-R (MINECO/FEDER).

REFERENCES

[1] J. A. Aguilar et al. 2007. The data acquisition system for the ANTARES neutrino telescope. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 570, 1 (2007), 107–116.

[2] Astigarraga, ME Pozo (on behalf of the ATLAS Collaboration). 2015. Evolution of the ATLAS trigger and data acquisition system. In *Journal of Physics: Conference Series*, Vol. 608. IOP Publishing, 012006.

[3] ATLAS Collaboration. 2008. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST* 3 (2008), S08003.

[4] ATLAS TDAQ Collaboration. 2016. The ATLAS Data Acquisition and High Level Trigger system. *Journal of Instrumentation* 11, 06 (2016), P06008.

[5] Tomasz Bawej, Ulf Behrens, James Branson, Olivier Chaze, Sergio Cittolin, Georgiana-Lavinia Darlea, Christian Deldicque, Marc Dobson, Aymeric Dupont, Samim Erhan, et al. 2015. The new CMS DAQ system for run-2 of the LHC. *IEEE Transactions on Nuclear Science* 62, 3 (2015), 1099–1103.

[6] Matías Bonaventura, Daniel Foguelman, and Rodrigo Castro. 2016. Discrete event modeling and simulation-driven engineering for the ATLAS data acquisition network. *Computing in Science & Engineering* 18, 3 (2016), 70–83.

[7] F Carena, W Carena, S Chapeland, V Chibante Barroso, F Costa, E Dénes, R Divià, U Fuchs, A Grigore, T Kiss, et al. 2014. The ALICE data acquisition system. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 741 (2014), 130–162.

[8] Tommaso Colombo, Holger Fröning, Pedro Javier García, and Wainer Vandelli. 2015. Modeling a Large Data-Acquisition Network in a Simulation Framework. In *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 809–816.

[9] Tommaso Colombo, Holger Fröning, Pedro Javier García, and Wainer Vandelli. 2016. Optimizing the data-collection time of a large-scale data-acquisition system through a simulation framework. *The Journal of Supercomputing* 72, 12 (2016), 4546–4572. https://doi.org/10.1007/s11227-016-1764-1

[10] Robert Cranfield, Piotr Golonka, Anna Kaczmarska, Krzysztof Korcyl, Jos Vermeulen, and Sarah Wheeler. 2004. Computer modeling the ATLAS Trigger/DAQ system performance. *IEEE Transactions on Nuclear Science* 51, 3 (2004), 532–538.

[11] M Frank, C Gaspar, B Jost, and N Neufeld. 2015. The LHCb Data Acquisition and High Level Trigger Processing Architecture. In *Journal of Physics: Conference Series*, Vol. 664. IOP Publishing, 082011.

[12] Karl Scott Hemmert, Mahesh Rajan, Robert J Hoekstra, Shawn LLNL Dawson, Manuel LANL Vigil, Daryl LANL Grunau, James LANL Lujan, David LANL Morton, Hai Ah LANL Nam, Paul Peltz Jr, et al. 2016. *Trinity: Architecture and Early Experience.* Technical Report. Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States).

[13] Pieter Hintjens. 2013. *ZeroMQ: messaging for many applications.* O'Reilly Media, Inc.

[14] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[15] Alejandro Santos, Pedro Javier García, Wainer Vandelli, and Holger Fröning. 2017. Modeling and Validating Time, Buffering, and Utilization of a Large-Scale, Real-Time Data Acquisition System. In *The Third International Workshop on Modeling and Simulation of Parallel and Distributed Systems (MSPDS 2017)*. IEEE, 519–525.

[16] Soo Ryu, on behalf of the ATLAS TDAQ Collaboration. 2017. FELIX: The new detector readout system for the ATLAS experiment. In *Journal of Physics: Conference Series*, Vol. 898. IOP Publishing, 032057.

[17] Andras Varga. 2010. OMNeT++. In *Modeling and Tools for Network Simulation*. Springer, 35–59.

[18] JC Vermeulen, S Hunt, C Hortnag, F Harris, A Erasov, RJ Dankers, and A Bogaerts. 1998. Discrete event simulation of the ATLAS second level trigger. *IEEE Transactions on Nuclear Science* 45, 4 (1998), 1989–1993.

[19] William Panduro Vazquez on behalf of the ATLAS Collaboration. 2017. The ATLAS Data Acquisition System in LHC Run 2. In *Journal of Physics: Conference Series*, Vol. 898. IOP Publishing, 032017.

[20] Yongwei Wu, Feng Ye, Kang Chen, and Weimin Zheng. 2014. Modeling of distributed file systems for practical performance analysis. *IEEE Transactions on Parallel and Distributed Systems* 25, 1 (2014), 156–166.

[21] Pedro Yebenes, Jesus Escudero-Sahuquillo, Pedro J Garcia, and Francisco J Quiles. 2013. Towards modeling interconnection networks of exascale systems with omnet++. In *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*. IEEE, 203–207.