# File-based data flow in the CMS Filter Farm

# File-based data flow in the CMS Filter Farm

**J-M Andre[5], A Andronidis[2], T Bawej[2], U Behrens[1], J Branson[4], O Chaze[2], S Cittolin[4], G-L Darlea[6], C Deldicque[2], M Dobson[2], A Dupont[2], S Erhan[3], D Gigi[2], F Glege[2], G Gomez-Ceballos[6], J Hegeman[2], A Holzner[4], R Jimenez-Estupiñán[2], L Masetti[2], F Meijers[2], E Meschi[2], R K Mommsen[5], S Morovic[2], C Nunez-Barranco-Fernandez[2], V O'Dell[5], L Orsini[2], C Paus[6], A Petrucci[2], M Pieri[4], A Racz[2], P Roberts[2], H Sakulin[2], C Schwick[2], B Stieger[2], K Sumorok[6], J Veverka[6], S Zaza[2] and P Zejdl[2]**

[1] DESY, Hamburg, Germany
[2] CERN, Geneva, Switzerland
[3] University of California, Los Angeles, California, USA
[4] University of California, San Diego, California, USA
[5] FNAL, Batavia, Illinois, USA
[6] Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

E-mail: `emilio.meschi@cern.ch`

**Abstract.**
During the LHC Long Shutdown 1, the CMS Data Acquisition system underwent a partial redesign to replace obsolete network equipment, use more homogeneous switching technologies, and prepare the ground for future upgrades of the detector front-ends. The software and hardware infrastructure to provide input, execute the High Level Trigger (HLT) algorithms and deal with output data transport and storage has also been redesigned to be completely file-based. This approach provides additional decoupling between the HLT algorithms and the input and output data flow. All the metadata needed for bookkeeping of the data flow and the HLT process lifetimes are also generated in the form of small "documents" using the JSON encoding, by either services in the flow of the HLT execution (for rates etc.) or watchdog processes. These "files" can remain memory-resident or be written to disk if they are to be used in another part of the system (e.g. for aggregation of output data). We discuss how this redesign improves the robustness and flexibility of the CMS DAQ and the performance of the system currently being commissioned for the LHC Run 2.

## 1. Introduction

The CMS experiment is one of two general purpose detectors located at the LHC at CERN, Switzerland. It is designed to study both proton-proton and heavy ion collisions at the TeV scale. A detailed description of the CMS can be found in [1]. It comprises about 50 million electronics channels and features a fully pipelined readout architecture, capable of operating at the machine crossing rate of 40 MHz without deadtime.

In the CMS Data Acquisition System (DAQ), signals from individual detector channels are held in low-level very front end analog or digital pipelines awaiting a trigger decision. A first-level trigger, accepting up to 100 kHz and consisting of custom electronic boards, operates on coarse-granularity, fast readout from the calorimeter and muon detectors to produce a trigger decision within about 1 $\mu$s. Raw data for accepted events are read out and assembled in two steps, using a complex of switched networks connecting the readout boards to a cluster of commercial computers running Linux. Three types of applications run in the cluster: the Readout Unit (RU) is connected to the detector front-end readout and performs a first data concentration stage, assembling event fragments from a given detector partition; the Builder Unit (BU) receives event fragments from the RUs and assembles full events which are then buffered while undergoing the final event selection in the Filter Unit (FU). The High Level Trigger (HLT) algorithms run within the FU and operate on full granularity event data from all subdetectors, selecting about 1-10% of the events accepted by the Level-1 trigger. The HLT uses offline-grade algorithms and the CMS offline reconstruction framework (CMSSW) [2]. Events accepted by the HLT are stored locally and subsequently transferred to the CERN computing centre for offline processing.

## 2. The upgraded CMS DAQ for Run 2 (DAQ2)

After successfully operating throughout the Run 1 of the LHC[8], the CMS DAQ underwent a major reimplementation. The main reasons for this are the replacement of equipment at the end of its life cycle, the need to replace obsolete technologies, the ever-increasing CPU requirements of the HLT (also related to the anticipated performance of the machine), demanding increased flexibility in integrating heterogeneous processor generations and architectures, and the requirement to accommodate new detector components and upgraded detector readouts exceeding the original DAQ specifications. While the general architecture remains the one described in Section 1, most of the networking and much of the software have been redesigned to take advantage of the new technologies and improve the robustness and maintainability of the system.

The first level of data concentration, which was based on Myrinet, has been reimplemented using 10/40 GE. The event builder network, which was based on Gigabit Ethernet [3], has been reimplemented using InfiniBand [4]. Since Gigabit Ethernet switch ports were relatively inexpensive at the time the DAQ1 was designed, each processing node was connected to the event-builder network and ran the event building (BU) locally, i.e. it combined the BU and FU functionality in each box. InfiniBand is 56 times more performant, but more expensive, so the BU and FU functionalities have been split in DAQ2 to efficiently exploit a relatively small InfiniBand network: 62 BU nodes are connected to the event builder network and each write complete events to a large RAMdisk of 240 GB. The choice of buffering event data on a RAMdisk is discussed in more depth in the following sections. More details on the DAQ2 event builder can be found in [5].

The hardware used to run the online selection in the DAQ1 (collectively referred to as Filter Farm) has evolved gradually, accommodating different generations of processors. The various generation of machines currently forming the filter farm, and their specifications, are summarised in table 1.

Some of the legacy processing nodes are not yet at end of life and had to be integrated in the DAQ2 with their GbE interfaces. On the other hand, recent dual CPU motherboards have enough cores to require more bandwidth than what GbE can provide, while 10GbE NICs have become sufficiently inexpensive. The Event Builder - Filter Farm interconnect is therefore built around 40 GbE switches with 10 GbE breakout. The legacy nodes are further connected in cascade to legacy 540-port 1/10 GbE switches. FU nodes are statically allocated to service a particular BU node and are treated as a unit with it, called an appliance. Different generations are allocated to different appliances in different numbers according to their relative performance (table 1). The system can thus remain evolutionary by allowing, e.g., to connect future high-performance very-many-cores systems to the event builder using 40GbE in a separate appliance.

[8] The Run 1 DAQ system will henceforth be referred to as “DAQ1”

**Table 1.** The Filter Farm currently consists of three generations of hardware. The legacy machines, which will be progressively retired during Run 2, will not be retrofitted with 10 GbE interfaces.

| Machine Type | Years oper. | CPU (2×) | Cores / node | RAM (GB) | Total nodes | Total cores | Cores / BU | HLT rel. perf. | Network |
|---|---|---|---|---|---|---|---|---|---|
| C6100 | 2011-15 | X5650 | 12 | 24 | 288 | 3456 | 216 | 0.6 | 1GbE |
| C6220 | 2012-16 | ES2670 | 16 | 32 | 256 | 4096 | 256 | 1.0 | 1GbE |
| s2600KP | 2015-19 | ES2680v3 | 24 | 64 | 360 | 8640 | 288 | 1.66 | 10GbE |
| Total | | | | | 904 | 16192 | | | |

Finally the DAQ1 data-logging system, a SAN solution based on JBOD cages connected over fibrechannel to individual nodes, and using ad-hoc software for data collection and storage [6], was replaced with state-of-the-art NL-SAS-based hardware and a cluster file system. The DAQ2 Storage and Transfer System (STS) fulfils the additional requirements of output bandwidth dictated by the new machine conditions and the experiment physics goals, while using a cluster filesystem radically reduces the amount of ad-hoc software and controls. More details on the DAQ2 STS can be found in [7].

## 3. File-based Filter Farm

In the DAQ1 system, the HLT algorithms ran inside the DAQ framework. A specific application embedded the reconstruction process and took care of control and data flow using inter-process communication [8]. The resulting tight coupling required several precautions: the different state models and services of the XDAQ framework [9] and CMSSW had to be aligned, as well as their release schedules; special care had to be taken when spawning HLT processes from XDAQ to avoid interactions between the two frameworks; the same compiler and external libraries versions had to be used. The File-based Filter Farm (FFF, figure 1) of the DAQ2 uses file systems to decouple the data flow among the various stages of processing. Using files as an interface among different processes is at the same time well proven and well adapted to the use with a reconstruction framework which is offline-oriented.

Each BU writes fully built events in files on the local RAMdisk. A RAMdisk-based system turned out to be the only solution which could provide sufficient bandwidth to meet the requirements, i.e. up to 2GB/s per BU concurrent input and output. The FU processors mount the RAMdisk from the BU over NFSv4, read input files from this disk and process them. NFSv4 uses TCP/IP and a stateful implementation of the NFS protocol, thus being relatively exempt from lock-ups and providing full support for remote locking, which is necessary for the arbitration of file access over several machines. After running the HLT algorithms, accepted events are written to the FU local disk in a format that allows combining them by concatenation. Successive stages of merging combine output from all the HLT processes in one appliance on the BU output disk. A final global merge combines output from all appliances in a cluster file system.

By separating the data-flow and the event processing, one expects to improve the resiliency of the system against localized failures, e.g. the loss of one processing node or a pathological event tripping the HLT. In this respect, using files for data flow has many advantages: a file system provides a simple mean of sharing a buffer, which comes with built-in resource accounting,

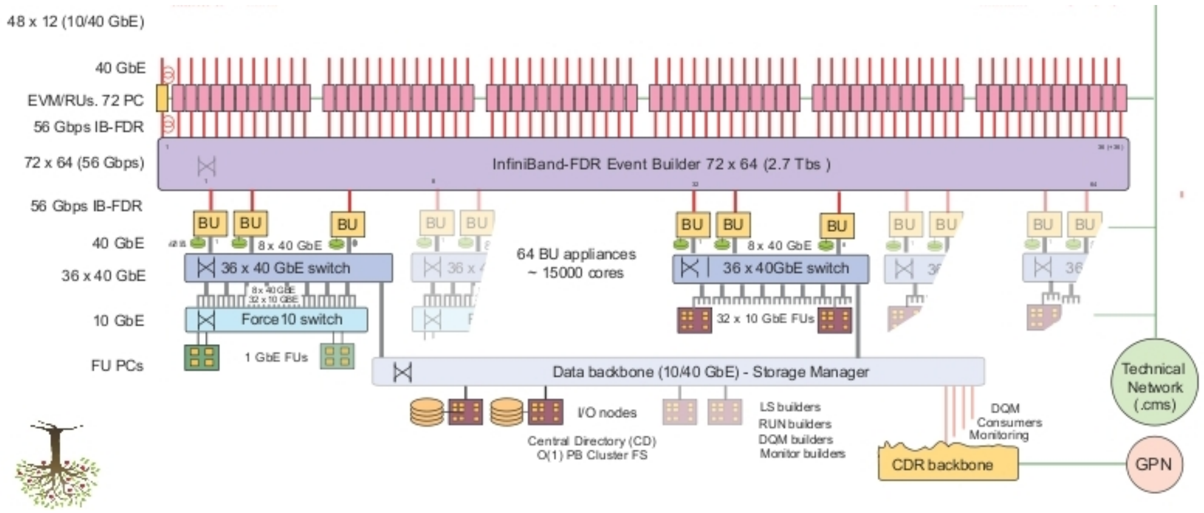

**Figure 1.** Schematic view of the DAQ2 Event Builder and the File-based Filter Farm components. Builder Units are connected to the event-builder network based on InfinBand, while the FU processing nodes connect to the BU via Ethernet. An appliance with legacy processors (left) uses an intermediate (legacy) 1/10 GbE switch, while recent ones have a 10 GbE connection. Both use a 40 GbE uplink to the BU.

arbitration and bookkeeping. A network filesystem provides the same advantages over different nodes while being in itself network-agnostic.

Taking advantage of the increased memory bus speeds and the relative low price of RAM, a buffer that allows significant time decoupling on the BU (order of minutes under normal running conditions) is possible, thus nicely accommodating the relatively long initialization times of CMSSW, due mainly to conditions loading. The time decoupling at all stages also improves efficiency at run boundaries, by allowing the asynchronous start of the next run while the processing of the previous one (HLT and/or output handling) is being completed [10].

## 4. Control and data flow

The FFF operates as a service and in an entirely data-driven fashion. At each run start, the BU creates a run directory containing the run number and the configuration for the HLT processes (trigger menu). The hlt daemon service (*hltd*), running on the FU, detects a new run directory and starts the HLT processes with the specified configuration. The service allocates CPU resources to a particular run by moving representative files from a pool of free resources to a specific directory: if the FU is still processing data from a previous run, or if resources on the FU are allocated for other tasks (e.g. for opportunistic offline processing), they are not made available to the new run until they are returned to the free pool. Likewise, if processors fail during operation and cannot be recovered, the corresponding resources are removed: the BU can thus modulate its input bandwidth based on the actual CPU available in the appliance, and generate backpressure if no resources remain.

Once the HLT processes are started, they look for input on the BU RAMdisk. The mechanism to arbitrate access to input files is discussed below.

### *4.1. Bookkeeping and arbitration*

In CMS, a Luminosity Section (LS) is defined as a quantum of data taking[9]. An LS is defined as a fixed time span lasting a predefined number of LHC orbits and treated as a unit. In the FFF this translates in the requirement that no input or output file can cross LS boundaries. At the same time, since output files from the HLT must be closed and assembled at the end of each LS, all input event for that LS must then be accounted for. In order to efficiently exploit CPU resources, every HLT process in the farm must receive sufficient input in an LS to work for the entire LS, and the minimum theoretical latency when the system is fully loaded (from the last input event for an LS to the last output file close) is the LS duration. A quick calculation shows that a single BU must generate of the order of 10 files per second to meet this requirement. The bookkeeping and the arbitration of access to this large number of files may seem a daunting task.

In order to maintain control over the data flow and be able to trace every single event at every stage in the system, every data file in the FFF is accompanied by a metadata file in JSON format, which is also used as a hook to indicate completion and ownership of the data. The metadata file for raw data from the event builder contains information about the number of events in the file and the originating BU. This way, the data flow control does not need access to the actual data files which are only accessed by CMSSW. When HLT processes in an appliance start working on a run, they look for input in the corresponding RAMdisk directory. Arbitration is achieved by means of a single index file, containing information about the next available input: every process tries to lock it and, upon success, reads the information and takes ownership of a file by moving the corresponding JSON metadata to its local directory. It then updates the index to the next file, releases the lock, and starts reading the input. With this technique, it is possible, for example, to reach a stable throughput of slightly over 3 GB/s with 256 processes in 8 machines concurrently accessing the BU RAMdisk over 10 GbE (figure 2 left). Locking is known to be a critical operation in network file systems. Therefore, it is important that the lock contention latency be under control under heavy demand. Figure 2 (right) shows the lock acquisition latency under the same conditions discussed above. In order to avoid wasting CPU cycles while processes are waiting for the lock, a double buffer is used for input files, such

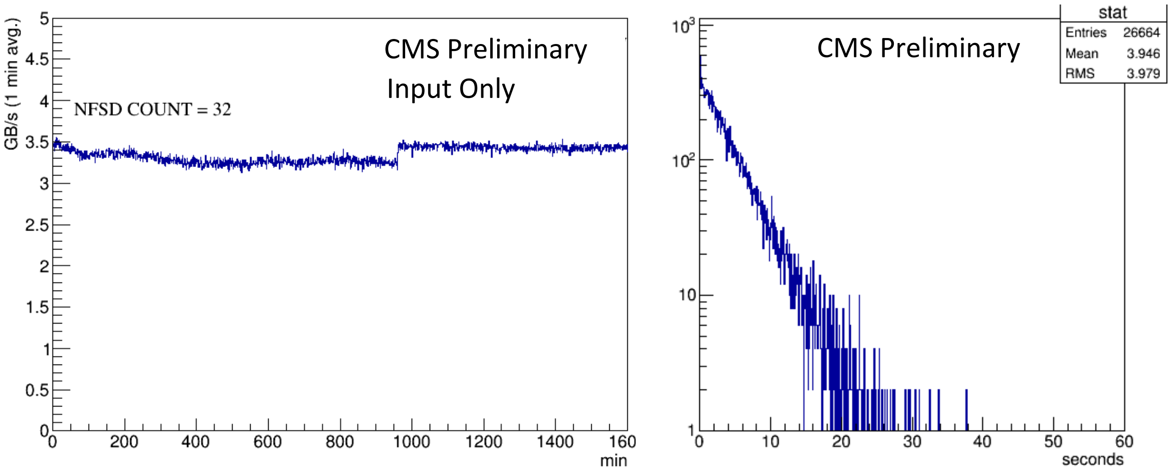

**Figure 2.** Input throughput from RAMdisk in an appliance with 8 nodes and 256 processes (left). Lock contention latency under the same conditions

[9] controlled by the Trigger Control and Distribution System (TCDS) and used later for the accounting of effective integrated luminosity

that a process never remains idle. This on the average only adds few seconds to the overall input-to-output latency.

### *4.2. Data and metadata streams*

The output of an HLT process is organized in "streams". Streams categorize the type of event data recorded (i.e. physics, calibration, data quality monitoring, etc.). Each HLT process therefore produces one file per LS and per stream, and a typical HLT configuration defines of order of 10 output data streams. Since, under normal conditions, the HLT accepts 1-10% of the events, the required output throughput per FU is only few MB. Calibration and monitoring streams produce together a similar amount of data, hence output can be comfortably written to the machine local disk. Each output file is accompanied by a metadata JSON file similar to the ones produced by the BU, which serves again as a completion indicator and contains all the information needed for bookkeeping (i.e. total number of processed and output events, originating process, etc.). The *hltd* service uses inotify[10] to detect the appearence of new output metadata files, and the information contained therein to verify that all input events in that particular FU for a particular LS and stream are accounted for. The corresponding output data files from the various processes are then concatenated (see above concerning the output format), and the metadata aggregated to create a single pair of data/metadata files which are copied back to the BU output disk over NFS. A chain of merger processes then takes care of moving and aggregating output data further [7]. The overall disk configuration and data traffic in an appliance are illustrated in figure 3.

The aggregated disk throughput required on the BU to receive output from the entire appliance under normal conditions is of the order of 50 MB/s, which can be provided by a standard array of magnetic disks. Figure 4 illustrates the performance of the BU output disk. In this example, the BU NFS server is concurrently loaded with full input traffic from 8 FUs in the appliance. The configuration with a 4-disk raid-0 array is shown to provide an ample margin on the output throughput required.

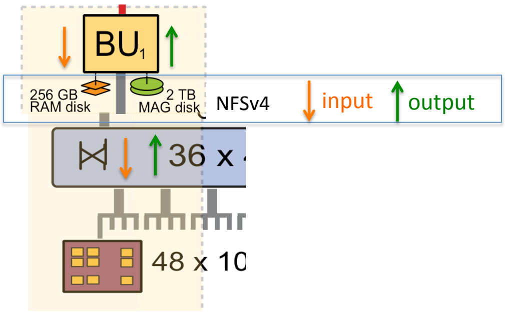

**Figure 3.** Detailed filesystem configuration and data traffic in the appliance

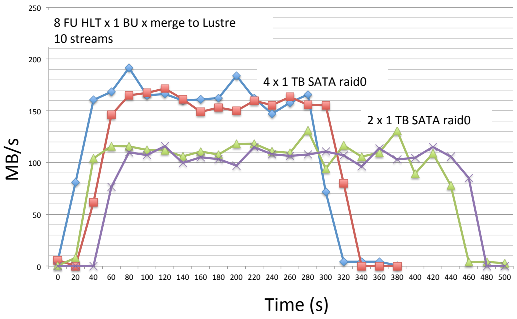

**Figure 4.** End-to-end output throughput for the appliance discussed above and for two different disk configurations.

Besides the event-data streams, the system can handle a diverse array of non-event data, mainly used for monitoring. These are generated as a byproduct of the algorithm execution and include:

- data quality histograms accumulated in the algorithms using the full Level-1 statistics
- counters of Level-1 and HLT accepts

[10] https://www.kernel.org/doc/Documentation/filesystems/inotify.txt

- event summary data used for quick feedback on potential interesting events

Non-event data streams are handled exactly like event data, and the resulting files used by specialized applications or migrated to a database by the STS.

### *4.3. Mechanisms of data flow control*

The main mechanism to control data flow in the FFF uses information on the available resources at the different stages of processing to modulate the data traffic on each appliance independently. For example, the BU calculates how many events per second it is going to build based on the available CPU in the appliance. Thresholds on the data buffers are applied at all stages to detect conditions where the appliance cannot keep up with the current data traffic and must be throttled. For example, when the RAMdisk reaches an almost-full threshold, the BU stops requesting events to the event builder. Building is not resumed until the RAMdisk usage goes below a safe threshold. Similarly, when the FU output buffer is almost full, the BU will be throttled. If safe running conditions are exceeded (e.g. not enough CPU, too high output rate) this will ultimately result in DAQ backpressure which will throttle the Level-1 trigger (figure 5).

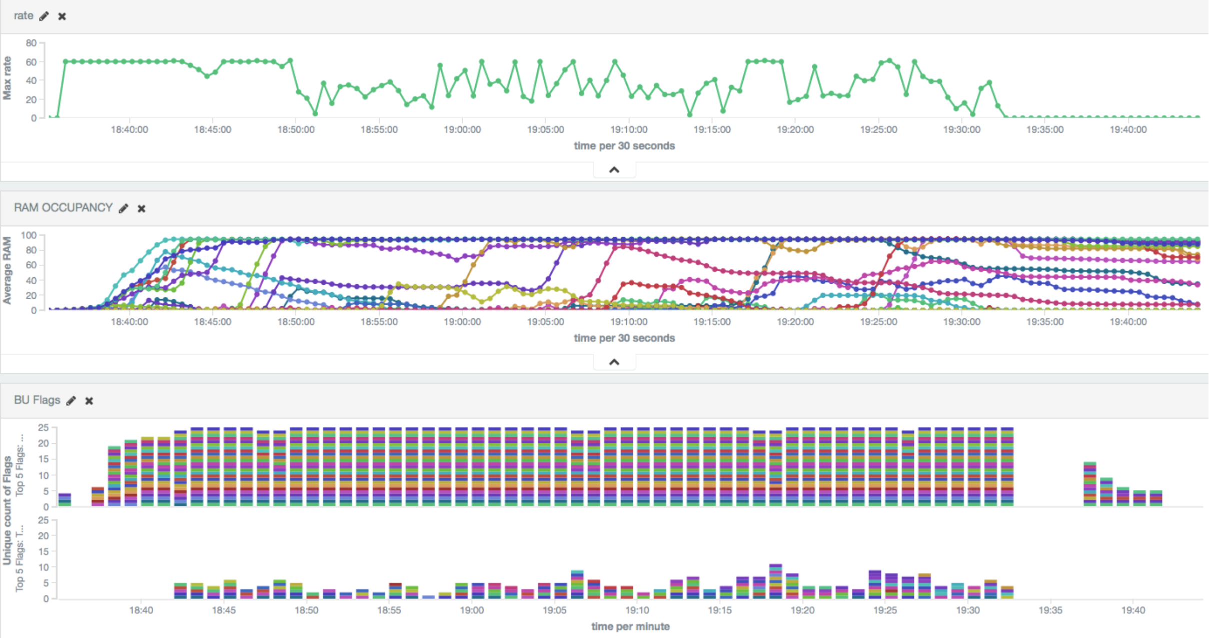

**Figure 5.** Illustration of the mechanism of backpressure as visualized by the FFF monitoring system. The top plot shows the input rate. As the system cannot keep up with the rate (in this case because of an HLT misconfiguration), the RAMdisk occupancy goes up on most BUs (middle plot). As the “almost-full” threshold is exceeded, most BUs raise the “throttle” flag (top part of bottom plot), and some of them raise the “blocked” flag (bottom part of bottom plot). As a consequence, the input is throttled and starts fluctuating around an acceptable level following the BU RAMdisk usage.

### *4.4. Data flow and execution monitoring*

The metadata generated at every stage of processing are constantly aggregated by the *hltd* service and used for data bookkeeping. They enable to account for every single input event all the way to individual output streams. The HLT is monitored by sampling the execution of the algorithms, thus providing a statistical estimate of their CPU usage. The *hltd* service also generates information about system parameters and available resources. Full-granularity and aggregated metadata files are injected in a monitoring system based on a distributed search engine, elasticsearch [11]. Details on the FFF monitoring system can be found in [12]. As an example, figure 5 shows an analysis of backpressure and RAMdisk occupancy on the system using elasticsearch.

## 5. Conclusions

The CMS File-based Filter Farm for Run 2 uses files and file systems for the control, monitoring, and data flow of the HLT and it is entirely data-driven. The system is now in production and performing as expected so far (though Run 2 has not properly started yet). Initial measurements on the production system indicate it will meet the specifications, i.e. input at the full L1-trigger accept rate of 100 kHz, up to 200 ms HLT CPU time per event, and order of 1 kHz aggregated event rate at the output. This architecture, using a conventional technology in a slightly unconventional way, is particularly suitable for a system which is in constant evolution and, being entirely data driven and otherwise similar to a standard batch farm, supports well the opportunistic use of online computing resources for offline tasks during periods of machine unavailability.

## Acknowledgments

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

## References

[1] Chatrchyan S *et al.* (CMS) 2008 *JINST* **3** S08004
[2] Lange D J (CMS) 2011 *Journal of Physics: Conference Series* **331** 032020
[3] Bauer G *et al.* 2008 *IEEE Trans.Nucl.Sci.* **55** 198–202
[4] *InfiniBand Trade Association* URL `http://www.infinibandta.com`
[5] Mommsen R K *et al. these proceedings*
[6] Bauer G *et al.* 2010 *J.Phys.Conf.Ser.* **219** 022038
[7] Darlea L G *et al. these proceedings*
[8] Spataru A *et al.* 2012 *Journal of Physics: Conference Series* **396** 012008
[9] Gutleber J *et al.* 2010 *Journal of Physics: Conference Series* **219** 022011
[10] Bauer G *et al.* 2014 *Journal of Physics: Conference Series* **513** 012025
[11] URL `https://www.elastic.co/`
[12] Morovic S *et al. these proceedings*