ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA177

STATUS OF THE DEVELOPMENT OF THE EXPERIMENT DATA ACQUISITION PIPELINE FOR THE EUROPEAN SPALLATION SOURCE*

A. H. C. Mukai†, M. J. Christensen, J. Nilsson, M. G. Shetty, T. S. Richter
European Spallation Source ERIC, Copenhagen, Denmark
M. Brambilla, M. Könnecke, D. Werder, Paul Scherrer Institut, Villigen, Switzerland
F. A. Akeroyd, M. J. Clarke, Science and Technology Facilities Council, Didcot, United Kingdom
M. D. Jones, Tessella, Abingdon, United Kingdom

Abstract

The European Spallation Source will produce more data than existing neutron facilities, due to higher accelerator power and to the fact that all data will be collected in event mode with no hardware veto. Detector data will be acquired and aggregated with metadata coming from sources such as sample environment, choppers and motion control. To aggregate data we will use Apache Kafka with FlatBuffers serialisation. A common schema repository defines the formats to be used by the data producers and consumers. The main consumers we are prototyping are a file writer for NeXus files and live reduction and visualisation via Mantid. A Jenkins-based setup using virtual machines is being used for integration tests, and physical servers are available in an integration laboratory alongside real hardware. We present the current status of the data acquisition pipeline and results from the testing and integration work going on at the ESS Data Management and Software Centre in collaboration with in-kind and BrightnESS partners.

INTRODUCTION

The European Spallation Source (ESS) is a spallation neutron source currently being built in Lund, Sweden. It will operate as a user facility offering a high brightness neutron beam in long pulses that can be tailored for adjusting resolution and bandwidth [1].

Located in Copenhagen, Denmark, the ESS Data Management and Software Centre (DMSC) is developing software for the experiment data acquisition pipeline in collaboration with the Science and Technology Facilities Council (STFC), as an in-kind partner, and Paul Scherrer Institut (PSI) and Elettra as part of the BrightnESS project [2]. This pipeline will transport and transform data from sources in the instrument, including neutron detectors and EPICS servers, to software performing tasks such as live experiment feedback and file writing. In the following sections, we discuss the architecture and components of the data acquisition pipeline, their current status and the tests being used to evaluate them, and present the conclusions and plans for future work.

* This work is partially funded by the European Union Framework Programme for Research and Innovation Horizon 2020, under grant agreement 676548.
† afonso.mukai@esss.se

Event Mode Acquisition

Instrument data at ESS will be acquired mainly in event mode, i.e., each neutron that is detected generates a pair of pixel identifier (ID) and timestamp values, where the ID corresponds to the location of the neutron in the detector. In this approach, in contrast with acquisition in histograms, the list of individual events is stored for subsequent experiment steps, such as data reduction and analysis; this allows the user to create histograms using different criteria, if desired. In addition, there will be no hardware veto for automatically stopping acquisition in case of chopper phase errors. Accelerator pulse information and the chopper top dead centre (TDC) signals will be acquired and attached to the datasets, allowing filtering to happen during the data reduction step.

Due to the high brightness beam and event mode acquisition, experiments at ESS will generate large volumes of data. Table 1 shows anticipated neutron event rates for some ESS instruments [3]. The proposed architecture for the data acquisition pipeline addresses these requirements with parallelisation and a clustered approach to data aggregation.

Table 1: Anticipated Neutron and Detector Rates for Some Early ESS Instruments

Instrument	Rate on Sample [n/s]	Rate on Detector [Hz]	Data Rate [MB/s]
BEER	10^9	2×10^5	1.6
C-SPEC	10^8	2×10^5	1.6
DREAMS	3.4×10^8	10^7	80
ESTIA		10^8	800
FREIA	5×10^8	1.2×10^7	96
HEIMDAL	2×10^9	8×10^6	64
LOKI	$\leq 10^9/cm^2$	4×10^7	320
SKADI	$\leq 10^9/cm^2$	4×10^7	320
T-REX	10^8	2×10^5	1.6

THE DATA ACQUISITION PIPELINE

ESS instruments will use an aggregator-based data acquisition pipeline. Apache Kafka was chosen as the central technology for aggregation and streaming, using Google FlatBuffers for serialisation. In this architecture, producers and consumers of data are decoupled and exchange data through the aggregator. Figure 1 shows an overview of the

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA177

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

main components of the architecture and the flow of data among them.

Data Aggregation and Streaming

Aggregation of the instrument data simplifies the design of consumers by allowing them to subscribe to data of interest using only one protocol (the Apache Kafka protocol, in this case). This decouples them from potentially different semantics on the data source side, such as control system or detector readout-specific protocols.

Apache Kafka The central piece of the aggregation and streaming infrastructure is Apache Kafka, an open-source distributed streaming platform [4]. It works as a clustered publish-subscribe messaging system and is designed for scalability and robustness to hardware failures.

When sending data to the cluster, a client publishes messages to a topic, which is identified by a string. Topic data are written to disk with configurable persistence, in one or more partitions that can be allocated to different brokers, allowing the system to scale by adding more servers to it. The C/C++ library librdkafka [5] has been chosen as the standard for clients to communicate with the Kafka cluster.

In this architecture, logic and protocols for dealing with each type of data source and destination are kept in the producer and consumer applications' code. The aggregator is a generic application that needs to be configured, but does not require dedicated development.

Different topics are used for each source of data at each instrument. A naming convention has been established as `<instrument>_<datatype>`, defining topic names such as `C-SPEC_detector` and `HEIMDAL_monitors`. High volume data topics, such as for neutron detectors, can have multiple partitions, with a different detector panel being assigned to a different partition if required for the cluster to handle the data rates. Events from a single pulse are combined in a message and sent to Kafka directly from the event formation software.

For lower data rate topics, such as those containing EPICS metadata, values from multiple sources are multiplexed over the same topic by adding both an identifier and the value to the messages. The EPICS to Kafka Forwarder is being developed as part of the pipeline to send these data to the Kafka cluster.

Kafka can also be used as a command bus, and dedicated topics have been created for control and status messages to be exchanged by applications, using JSON. These messages are currently used for commands like starting and stopping EPICS data forwarding and file writing, and also to send configuration to some of the applications at runtime.

FlatBuffers and Streaming Data Types For data to be successfully transferred between applications, a common format has to be adopted. Kafka does not process the contents of the messages it receives, so the agreement only needs to be established between the producing and consuming actors. This is done using Google FlatBuffers, an efficient open source serialisation library [6]. It includes a compiler that takes a schema as input and produces a programming language-specific file for inclusion by projects using that schema, e.g. a C++ header file or a Python module.

A set of common schemas have been defined and are kept in a source code repository [7]. Schemas are identified by a 32-bit value that is sent in every serialised buffer. This allows receivers to determine the correct schema for unpacking; the adopted convention for the repository is to prefix the schema file name with that identifier. It is possible to make backward compatible modifications to a schema which is already in use and proposed changes to released schemas have to undergo a review process by another project member before being accepted. Table 2 shows the main schemas in use by the pipeline applications.

Table 2: FlatBuffers Schemas

Schema	Description
ev42_events.fbs	Neutron detection event data
f142_logdata.fbs	Scalar and array data
f143_general.fbs	General EPICS structure

Data Producers

Data sources are instrument components such as neutron detectors, choppers, motors and sample environments. All of the time-sentsive data and metadata will be timestamped at the origin using the ESS distributed timing system [8].

Event Formation Before neutron event data is fed into the relevant Kafka topic the acquisition pipeline interface to the detectors will receive raw event data over UDP. These Event Formation Units (EFUs) will have a direct fibre connection to detector panels and process their output to generate the detector ID and timestamp pairs. Each type of detector requires specialised event formation algorithms. The processing happens in software, allowing different methods to be easily prototyped and tested before the system goes into production [9, 10]. The EFU design is modular and consists of separate input, processing and output threads, which allows a common software infrastructure that performs network input and output to Kafka to be reused for the various detectors.

In instruments with multiple detector panels, multiple EFUs can be used, with each connecting to a different panel in parallel; this allows the total neutron event throughput to scale horizontally by adding more units. Each of the EFUs can write to a different Kafka partition of the same topic. This is illustrated in Fig. 2. The Kafka client that sends the events to the cluster is directly integrated into the output thread and uses the *ev42_events.fbs* schema. The EFU can send metrics to Graphite and log messages to Graylog.

EPICS Forwarder Non-neutron metadata will arrive through EPICS [11] from sources such as the instrument

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA177

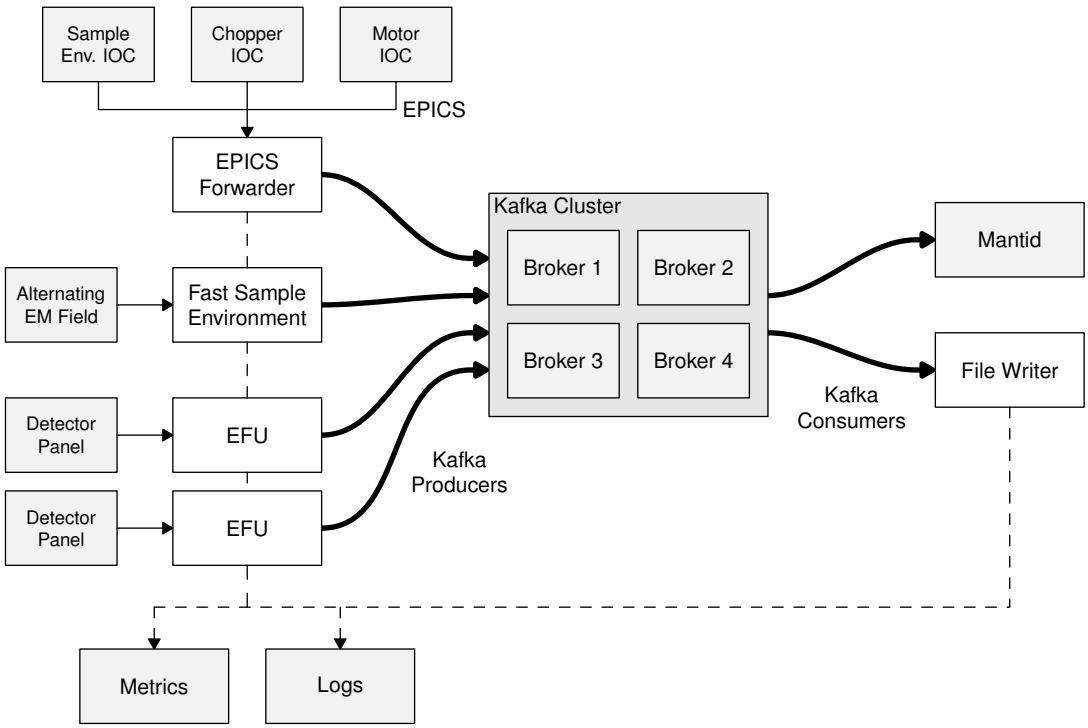

Figure 1: System architecture and data flow. The components in dark are being developed as part of the pipeline; dashed lines represent metrics and log messages.

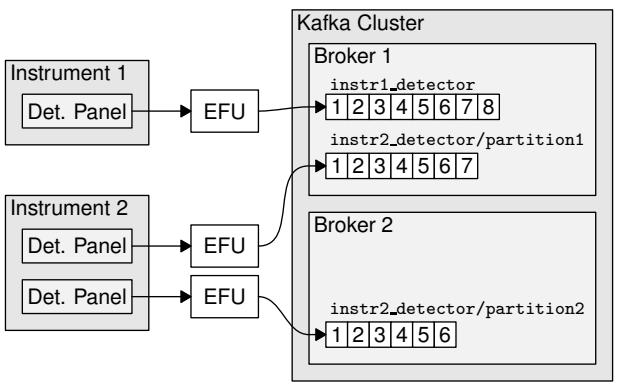

Figure 2: The output of each detector panel is processed by a different EFU, which, in turn, can send messages with events to different Kafka topic partitions.

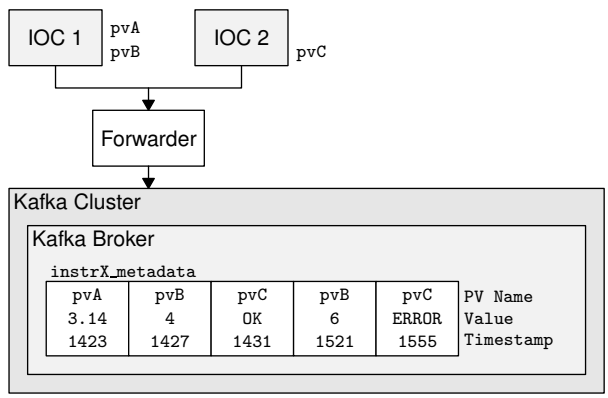

Figure 3: EPICS data is forwarded to the Kafka cluster, with different PV updates multiplexed over the same topic.

choppers, motors and sample environments. These components expose setpoint and readback values as process variables (PVs). The EPICS to Kafka Forwarder is being developed for monitoring PVs of interest and forwarding their values to the Kafka cluster [12, 13]. PV values are serialised and sent to the Kafka cluster multiplexed over a dedicated data topic, as illustrated in Fig. 3. The *f142_logdata.fbs* and *f143_general.fbs* schemas are currently used by the Forwarder.

Fast Sample Environment Measurements of alternating electric and magnetic fields, and some strain and pressure sensors are also expected to generate data at high rates, with anticipated frequencies on the order of 1 kHz to 1 MHz. Possibilities for handling these data include using EPICS and the Forwarder, a directly integrated Kafka producer, or sending data to an EFU.

Data Consumers

Consumers of experiment data subscribe to the Kafka topics of their interest, using the associated FlatBuffers schemas to deserialise the aggregated data. The two main consumers of the aggregated data currently being considered are the NeXus File Writer and the live data reduction and visualisation system.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA177

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Most consumers are expected to consume data in real time, as aggregation occurs. Consumers can, however, fetch older messages from the Kafka cluster, provided they are still available in its storage. The cluster retention parameters can be tuned, to ensure persistence is long enough for consumers' needs, considering latencies in the system, possible failures and available storage volume.

NeXus File Writer Experiment data will be persisted to files conforming to the NeXus standard [14] for offline reduction, analysis and archiving. For that, a NeXus File Writer is being developed as part of the data acquisition pipeline [12,15]. This application subscribes to the complete set of Kafka topics containing neutron detector events and metadata for one instrument, deserialises the FlatBuffers messages and writes data to files according to a specified hierarchy and naming scheme. As the File Writer writes both event data and metadata, it uses most schemas that can be used by the producers and, in the case of EPICS metadata, also demultiplexes the values from the common data topics used by the Forwarder.

Live Data Reduction and Visualisation Mantid [16] will be used for data reduction at ESS and work is in progress at DMSC and partner institutes to adapt it to consume data from Kafka in real time. This will allow it to be used to provide live visualisation and feedback for experiments.

STATUS OF THE PIPELINE

Development of the pipeline and its components is progressing as planned. The EFU, the EPICS Forwarder and the NeXus File Writer are being actively developed and are capable of processing and transporting data through the pipeline. Apache Kafka has been deployed to different environments on physical as well as virtual servers, for development and testing.

Metrics and Logging

Graphite [17] is currently being used for storing metrics. It includes a server that receives metrics from a network connection and stores them in a time series database. Grafana [18] is used for the visualisation. In the packaging step for the installation of Kafka, the kafka-graphite [19] library is added to the package for sending broker metrics to Graphite. This provides a simple way to monitor rates of data being sent to and retrieved from each broker, and also to obtain rate metrics for specific topics.

As the pipeline consists of a number of distributed components that will run on different machines, a centralised log solution has been adopted. Log messages are sent to a Graylog server [20], using the graylog-logger library developed at DMSC [21].

Building and Testing

Each commit to a repository triggers an automated build job, that compiles the software and runs existing tests. Besides this, an automated integration test is run twice a day, fetching the latest version of each application from the master branch build, deploying them to a set of virtual machines and running a series of commands and checks. Build and test automation are done using Jenkins [22].

Ansible [23] is used for automating the deployment and orchestrating the test steps. The build and test machines are Linux nodes running CentOS, with their configuration also kept in Ansible files. RPMs are generated for the installation of external components and libraries, such as Kafka and FlatBuffers.

After deploying the applications to the target machines and running simple tests to ensure all the required external services are running, the Ansible scripts start the applications and a simulation data streamer, which reads data from files recorded at test runs of the MultiGrid detector and sends them to the EFU. The EFU, in turn, processes the raw data and sends the calculated events to the Kafka cluster, from which the NeXus File Writer subscribes. In parallel, an EPICS IOC provides simulated data for the PSI SINQ AMOR instrument [12, 24]; the values of one of the PVs is sent to the Kafka cluster by the EPICS to Kafka Forwarder, and some updates to its value are performed. Metrics and log messages are sent to Graphite and Graylog and, at the end of the test, applications are stopped and the generated NeXus file is copied for a check of some of the event values. Figure 4 shows a Grafana screen with metrics from the Kafka cluster collected during runs of the integration test.

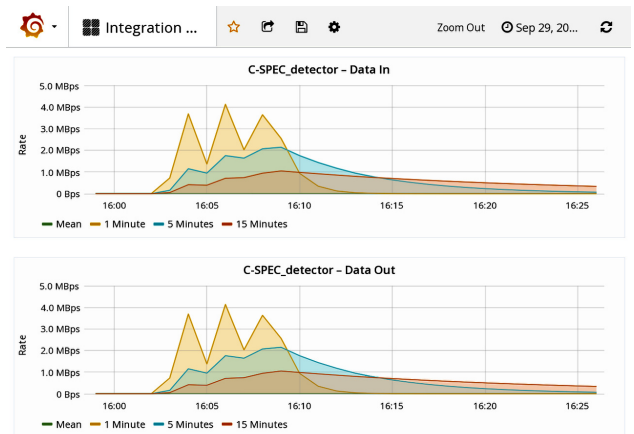

Figure 4: Grafana screen with Graphite metrics for the integration test Kafka cluster.

Dedicated hardware for testing and integration is available at the ESS Instrument Integration Project (ESSIIP) laboratory. Three high performance servers with two 10 Gbit Ethernet network cards each are currently located there, where other hardware such as choppers and sample environment equipment is also available. This allows for testing the complete pipeline in a pre-production, instrument-like setup, both for integration and performance.

Work is going on to improve reproducibility of the automated software builds in Jenkins by executing them inside Docker [25] containers. Docker images have been created for the build environments and a new container is started for

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUPHA177

each build, which then proceeds from a well known state and destroys the container after its end. The Conan C++ package manager [26] is being used to install dependencies inside the containers at the start of each build. Package recipes are kept in a public repository [27], while a local Conan repository in the build and test environment keeps binary versions of the packages for download by the build jobs. Figure 5 illustrates this workflow.

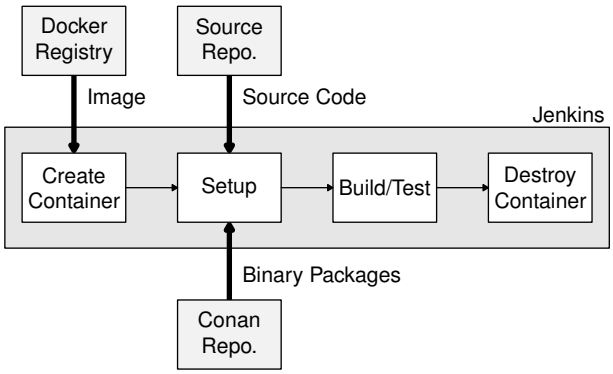

Figure 5: Jenkins build job workflow using Docker containers and a local Conan package repository.

CONCLUSION AND FUTURE WORK

The data acquisition pipeline for the ESS instruments is based on an aggregation and streaming architecture using Apache Kafka and Google FlatBuffers, with software developed at DMSC and partner institutes for event formation, EPICS forwarding and file writing. Development of the pipeline components is progressing as planned, with regular automated tests being performed individually and in an integrated setup.

Planned future work includes continuing the development of pipeline components and their integration with the experiment control program. The existing builds and tests will be improved by adding more checks and better reproducibility, with automated test environment configuration. Performance tests on physical machines will also be performed to evaluate the pipeline and tune the Kafka cluster configuration.

REFERENCES

[1] S. Peggs *et al.*, "ESS Technical Design Report", European Spallation Source, Lund, Sweden, ESS-doc-274, Apr. 2013.

[2] BrightnESS, https://brightness.esss.se

[3] A. Mukai, T. Richter, "Design report data aggregator software", European Spallation Source ERIC, Copenhagen, Denmark, BrightnESS Deliverable Rep. 5.1, Aug. 2016.

[4] Apache Kafka, http://kafka.apache.org

[5] librdkafka, https://github.com/edenhill/librdkafka

[6] Google FlatBuffers, http://google.github.io/flatbuffers/.

[7] Streaming Data Types, https://github.com/ess-dmsc/streaming-data-types

[8] J. Cereijo Garcia, T. Korhonen, J. H. Lee, D. Piso, and R. R. Osorio, "Timing System at ESS", in *Proc. 8th Int. Particle Accelerator Conf. (IPAC'17)*, Copenhagen, Denmark, May 2017, paper THPVA064, pp. 4588–4590.

[9] M. Shetty and M. J. Christensen, "Processing choices for detector types", European Spallation Source ERIC, Copenhagen, Den-mark, BrightnESS Deliverable Rep. 5.2, May 2017.

[10] Event Formation Unit, https://github.com/ess-dmsc/event-formation-unit

[11] Experimental Physics and Industrial Control System (EPICS), http://www.aps.anl.gov/epics/.

[12] D. Werder *et al.*, "EPICS Data Streaming and HDF File Writing for ESS Benchmarked Using the Virtual AMOR Instrument", presented at the 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17), Barcelona, Spain, Oct. 2017, paper THPHA167, this conference.

[13] Forward EPICS to Kafka, https://github.com/ess-dmsc/forward-epics-to-kafka

[14] NeXus, http://www.nexusformat.org

[15] Kafka to NeXus, https://github.com/ess-dmsc/kafka-to-nexus

[16] O. Arnold *et al.*, "Mantid–Data analysis and visualization package for neutron scattering and μSR experiments", *Nucl. Instr. Meth. Phys. Res. A*, vol. 764, pp. 156–166, 2014.

[17] Graphite, http://graphite.readthedocs.io/en/latest/.

[18] Grafana, https://grafana.com

[19] kafka-graphite, https://github.com/damienclaveau/kafka-graphite

[20] Graylog, https://www.graylog.org

[21] graylog-logger, https://github.com/ess-dmsc/graylog-logger

[22] Jenkins, https://jenkins.io

[23] Ansible, https://www.ansible.com

[24] sinq-amorsim, https://github.com/ess-dmsc/sinq-amorsim

[25] Docker, https://www.docker.com

[26] Conan, https://conan.io

[27] ess-dmsc Conan package repository, https://bintray.com/ess-dmsc/conan

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2017). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.