# BrightnESS

## Building a Research Infrastructure And Synergies For Highest Scientific Impact On ESS

## H2020-INFRADEV-1-2015-1

## Grant Agreement Number: 676548

## Deliverable Report: D5.5 Data aggregator software

# 1 Project Deliverable Information Sheet

| | | |
|---|---|---|
| **BrightnESS Project** | Project Ref. No. 676548 | |
| | Project Title: BrightnESS - Building a Research Infrastructure And Synergies For Highest Scientific Impact On ESS | |
| | Project Website: https://brightness.esss.se | |
| | Deliverable No.: 5.5 | |
| | Deliverable Type: Report | |
| | Dissemination Level: Public | Contractual Delivery Date: 31.07.2018 |
| | | Actual Delivery Date: 14.08.2018 |
| | EC Project Officer: Mina Koleva | |

# 2 Document Control Sheet

| | | |
|---|---|---|
| Document | Title: BrightnESS_Deliverable_5.5 | |
| | Version: 0.1 | |
| | Available at: https://brightness.esss.se | |
| | Files: 1 | |
| Authorship | Written by | Afonso Mukai (WP5) |
| | Contributors | Carlos Reis (WP5), Dominik Werder (WP5), Michele Brambilla (WP5), Mark Könnecke (WP5) |
| | Reviewed by | Tobias Richter (WP5 leader), Anne-Charlotte Joubert (WP1 project coordinator) |
| | Approved by | BrightnESS Steering Board |

# 3 List of Abbreviations

| | |
|---|---|
| API | Application Programming Interface |
| DM | Data Management (DMSC group) |
| DMSC | Data Management and Software Centre (ESS) |
| DOI | Digital Object Identifier |
| ECP | Experiment Control Program |
| EFU | Event Formation Unit |
| EPICS | Experimental Physics and Industrial Control System |
| ESS | European Spallation Source |
| HZB | Helmholtz-Zentrum Berlin (Germany) |
| IOC | Input/Output Controller |
| JSON | JavaScript Object Notation |
| MPI | Message Passing Interface |
| PSI | Paul Scherrer Institut (Switzerland) |
| PV | Process Variable (EPICS) |
| STFC | Science and Technology Facilities Council (UK) |
| SWMR | Single-Writer/Multiple-Reader (HDF5) |
| WP | Work Package |

# 4 List of Figures

## Table of Contents



## 5 Executive Summary

This document concerns BrightnESS Deliverable 5.5: “Data aggregator software”: a software system for data aggregation and streaming that has been developed for ESS neutron instruments. The system is based on a distributed producer-consumer architecture, where data is exchanged through Apache Kafka with FlatBuffers serialisation. The EPICS to Kafka Forwarder connects EPICS data sources to the aggregation system, while the NeXus File Writer generates files for subsequent data reduction and analysis. Both were developed as part of this BrightnESS task. Other software components developed as part of Work Package 5 and in collaboration with in-kind partners, such as the Event Formation Unit, the Fast Sample Environment and the Experiment Control Program NICOS, are integrated in this architecture.

Different types of tests have been performed and have confirmed that the system provides the functionality expected from it. Besides the automated unit and integration tests, demonstrations and deployments to laboratories and working neutron facilities showed the data aggregation and streaming system works well under production conditions. Performance tests were carried out and a good scaling behaviour was observed for Apache Kafka and the file writer, sufficient to deal with the high data rates expected from ESS instruments.

Recent activities include a quality improvement initiative, which has produced positive results on software design, documentation, maintainability and readability. A package manager has been adopted for management of dependencies to internal code and to third-party libraries. The combined build and test system has been enhanced through further automation and use of containerisation for build isolation and reproducibility.

The streaming framework will continue to be under active development after the end of BrightnESS, as ESS enters its commissioning and operations phases and partners continue with their deployments at their facilities. This demonstrates the fruitful completion of this task and work package as both the maintenance effort and the benefits of having a state-of-the-art open source data streaming framework are shared between a number of European sites.

# 6 Report on Implementation Process and Status of Deliverable

The data aggregation task is part of BrightnESS Work Package 5 ("Real-Time Management of ESS Data") and is being undertaken by the BrightnESS partners at the Data Management group (DM) at DMSC, Copenhagen University, Elettra in Italy and the Paul Scherrer Institut (PSI) in Switzerland. The DM group also has an in-kind collaboration agreement with the Science and Technology Facilities Council (STFC) in the United Kingdom for data streaming tasks.

This deliverable is the data aggregation software system which comprises open source software components both from third-parties and custom developments. The latter have been made available via Digital Object Identifiers (DOIs) as part of the BrightnESS MS35 milestone [1], [2]. This document briefly reviews the system architecture and presents the progress since the last deliverable [3], including updates to the forwarder and file writer software, the code quality initiative and work on tests, followed by a section on outstanding problems and risks and, finally, the conclusions.

# 7 Technical Content

Neutron instrument data acquisition at ESS will happen mostly in event mode, i.e. each detected neutron will generate an identifier-timestamp pair. Instrument metadata such as sample environment and motion control measurements will also be timestamped at their source, so that all data can be considered as a stream of events. The high neutron beam brightness results in high data rates from the detectors. This sets requirements that demand a scalable acquisition solution. An aggregation and streaming-based software architecture has been selected to address them [3]. The majority of the technical detail can be found in the recently submitted paper [4], that is part of this deliverable as appendix A. For that reason, the sections below only cover the architecture and the components briefly. Otherwise the text here focusses on some of the processes to achieve the results, which are not mentioned in the paper.

## 7.1 Data Aggregation and Streaming Architecture Overview

In the centre of the aggregation and streaming architecture is Apache Kafka, an open source distributed publish-subscribe streaming platform [5]. It provides a rich set of functionalities that allow producers to send data using named topics to which consumers can subscribe. Kafka brokers can be grouped in a cluster to provide reliability and scalability via topic replication and partitioning. Data is retained on disk in a configurable manner, either for a minimum time interval or until a set storage limit is reached.

The sources of data in the system include *Event Formation Units* (EFUs), the Fast Sample Environment and data from instrument EPICS *Input/Output Controllers* (IOCs). Detector readout electronics will provide raw data that the EFU software will receive and process to extract neutron events [6]. Fast electric and magnetic fields generate *Fast Sample*

*Environment* data [7]. IOCs running the *Experimental Physics and Industrial Control System* (EPICS) [8] will provide lower-rate metadata from choppers, motion control systems, equipment monitoring the sample environment, and other devices. An *EPICS to Kafka Forwarder* application has been developed to send these data to the Kafka cluster as part of this work package [1].

On the consumer side, the *NeXus File Writer* [2], also developed in this work package, subscribes to data from the Kafka cluster and writes them to files in the NeXus format [9], at the same time *Mantid* [10] will subscribe to data from Kafka for live reduction and visualisation. The file writer and the forwarder receive commands and configuration from the *Experiment Control Program* (ECP) *NICOS* [11] via Kafka, that also interacts with the instrument IOCs via EPICS process variables (PVs). Figure 1 presents an overview of the system architecture.

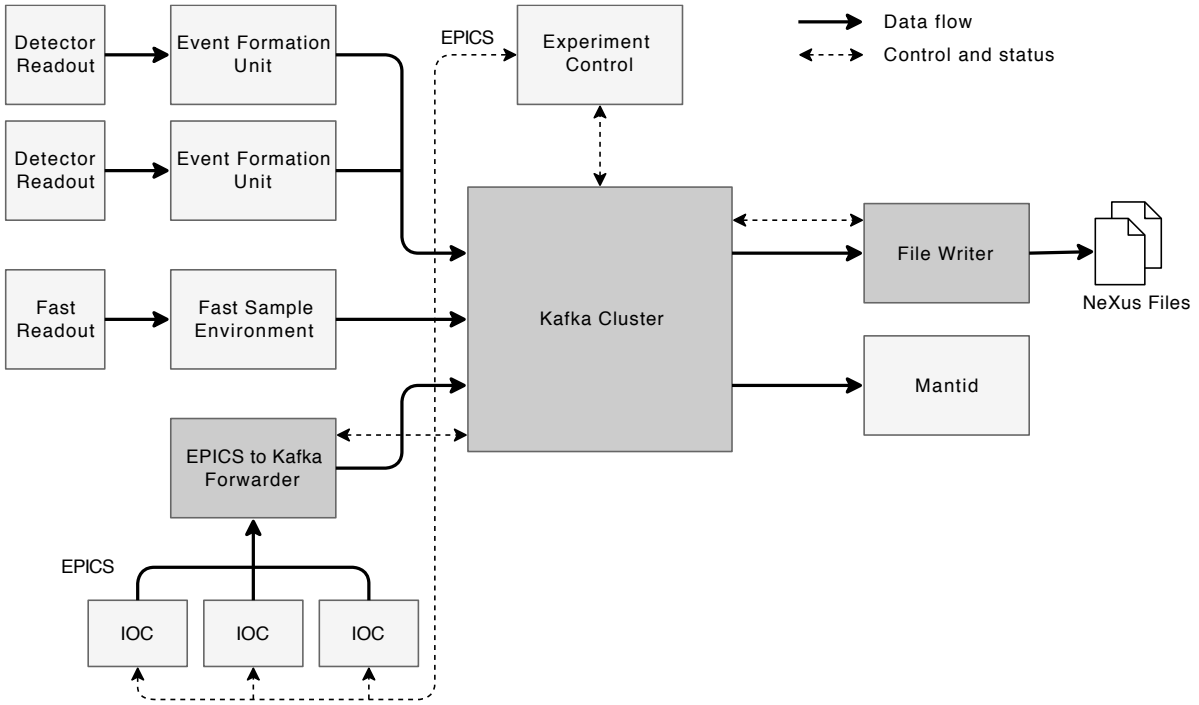


**Figure 1. Data aggregation and streaming system architecture overview. Components in dark grey are the focus of this work package.**

For Apache Kafka, messages are just binary blobs. Thus, in order for the different components of the streaming system to agree on the interpretation of the exchanged data and keep message size smaller, a common schema-based serialisation format for messages is required. The open source Google FlatBuffers library [12] has been selected as the serialisation technology for data aggregation and streaming. It provides a compiler that can generate language-specific files from schemas for inclusion by other projects. These generated files contain functionality to serialise and deserialise data that is exchanged through the Kafka cluster.

A number of schemas for neutron event streaming have been developed as part of the project. These schemas are kept in a version-controlled repository [13] and are named following a convention that includes a 4-character schema identifier as a prefix in the file name. That identifier is also included in the sent serialised buffers at a known offset, allowing data consumers to identify the schemas of received buffers at runtime.

## 7.2 Integration Milestone


**Figure 2. Group photo from the February 2018 integration meeting at Elettra.**

In February 2018, the BrightnESS WP5 Second Integration Meeting was held at the Elettra Synchrotron in Trieste, Italy. During that meeting, the activities in the BrightnESS Work Package (WP) 5 were discussed in sessions covering themes such as updates, risks, quality and integration. A series of tasks were identified for preparation for the final WP5 milestone and deliverables; these were written into tickets in a JIRA issue tracker [14] and collected under an integration milestone. The following are of relevance to the data aggregation and streaming task:

- Improve file writer and forwarder documentation
- Ensure EPICS PV values are forwarded when variables are not updated during the forwarder's running period
- Improve integration between the experiment control program and the file writer
- Ensure file writer instances only respond to commands addressed specifically to them
- Ensure the file writer does not unintentionally overwrite files
- Improve file writer error reporting on configuration file problems
- Have the file writer flag data written out of order
- Improve automation of the integration test environment
- Improve result checking and verification in integration test
- Perform stability tests for aggregation and streaming
- Run aggregation and streaming performance test

Besides the task-specific activities, a comprehensive code quality initiative was planned. In the sections below, the results of these and other related activities are described.

## 7.3 EPICS Metadata Forwarding Updates

The EPICS to Kafka Forwarder application uses JSON for configuration, command messages and internal data. The software was refactored to use the JSON for Modern C++ library [14] which reduced the overall code size, improved readability and reduced the risk for errors while increasing maintainability. Further refactoring included the updating of the command line interface of the application to use CLI11 [15]. As described in more detail in section 7.6.2, the forwarder now uses Conan [16] for reproducible, easy to perform builds with automatic dependency management on all supported platforms

In order to improve the interaction of the EPICS Forwarder with the other components of the system, the output of the status messages now contains more detailed information adapted to the needs of the experiment control program and system monitoring tools.

Most of the EPICS Forwarder's responsibility concerns communication, which provides challenges to testing. Unit testing was improved by using mocks and dependency injection to simulate and verify the communication for specific test cases. Figure 3**Error! Reference source not found.** shows the increase in the number of automated unit tests since March 2017.

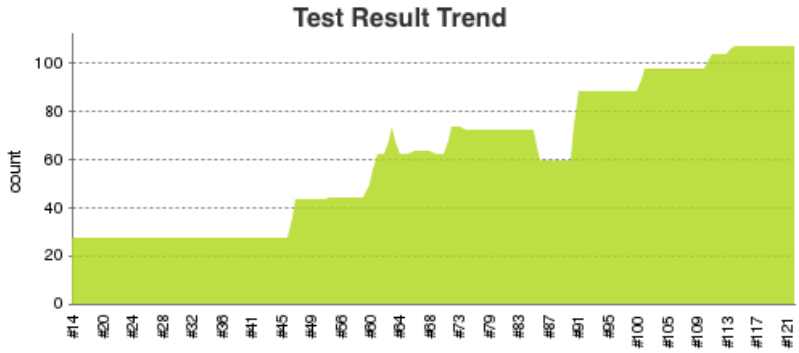


**Figure 3. Number of EPICS forwarder automated unit tests against build number since March 2017.**

Application-specific integration tests were added for testing the forwarder embedded in the overall architecture. The integration tests have been expanded and automated, such that they are run in a Docker-based container setup where all components of the infrastructure are assembled from the latest build artefacts in a clean and reproducible way.

Work on code quality yielded improvements such as treating compiler warnings as errors and improving the documentation of the software. The LLVM coding standard [17] has been adopted with a few minor changes to improve coherence among the ESS software components. For more details on code quality improvement, see section 7.5.

As an outcome of the code reviews, some classes of the code base have been refactored, where the grown functionality threatened to violate the single responsibility principle. A feature was added to publish the last known value of an otherwise idle EPICS process variable to the Kafka stream at configured intervals. This ensures that PV values are being written to the NeXus file even if they are not being updated during an experiment, which is normal for a number of parameters.

## 7.4 File Writing Updates

Writing of NeXus files is triggered by commands that are received by the NeXus File Writer through the Kafka cluster. These commands are in JSON format and contain the necessary information both to create the hierarchical structure of the HDF5 files, and to locate the data streams and corresponding topics on the Kafka cluster which are used to populate the file. The format of the JSON command has been changed to be more concise in the way that it allows the dynamic streams to be defined in line with the static structure. This change also made the definition of dynamic and static datasets more homogeneous.

Handling of string data types has also been improved and is now supported on static and dynamic datasets. Furthermore, HDF5 attributes can be specified for dynamic datasets and writer modules can use a reference implementation, which covers the most common case where attributes containing the datasets of the dynamic data stream are placed in an HDF5 group.

The write command now can also specify the time range of data to be written to file. This functionality allows writing NeXus files from data which is already persisted in the Kafka cluster storage and can be used to create HDF5 files even after the experiment, or re-create files with a different structure, streams or settings. The global configuration file allows specifying the default path for the output of HDF5 files.

Given a sufficiently fast file system, the write throughput of the file writer will at some point be limited by the performance of the single-process HDF5 library. For the ESS, this performance may not be sufficient to support all neutron detectors. In order to achieve higher throughput, the HDF5 library offers an MPI-I/O-enabled parallel version. Leveraging the full power of this approach requires the file writer to run as a multi-process MPI application. The performance of this method has been presented in [18]. In order to enable writing via MPI-I/O, the structure of the HDF5 file has to be created in advance. During MPI-I/O enabled writing it is not possible to create datasets or change their structure. The JSON command that triggers file writing therefore has to contain the full structure of the HDF5 file. As a result, optional type inference from the data actually received via the dynamic stream is no longer supported.

The file writer now supports writing using HDF5's Single-Writer/Multiple-Reader mode (SWMR). This allows the file to be read while it is being written and through the journaling required for SWMR also ensures that the file remains readable after a crash of the writer for whatever reason. In this mode, the structure of the HDF5 file also has to be fully created before writing of the actual data starts. This change is shared with the MPI-I/O approach of writing.

External dependencies have been updated and their number reduced. The JSON for Modern C++ library [14] has been adopted for handling JSON data in the application, providing adequate performance and a more usable application programming interface (API), which results in improved maintainability of the code base. For a cleaner and more extensible command line interface CLI11 [15] is now used. The HDF5 library provides an API that primarily targets the C language; a C++ API exists, but it does not leverage the power of C++. Instead, the new h5cpp C++ API for HDF5 [19] is now being used. For more interoperability, the status reporting has undergone some changes as well: individual Streamer instances now report their status and statistics about their performance to a Kafka topic.

As described in more detail in section 7.5, code quality in general has been improved. The changes include treating compiler warnings as errors, improvements to the documentation and using Conan to simplify building of the application (see section 7.6.2).**Error! Reference source not found.** Figure 4 shows the increase in the number of automated unit tests since November 2017.

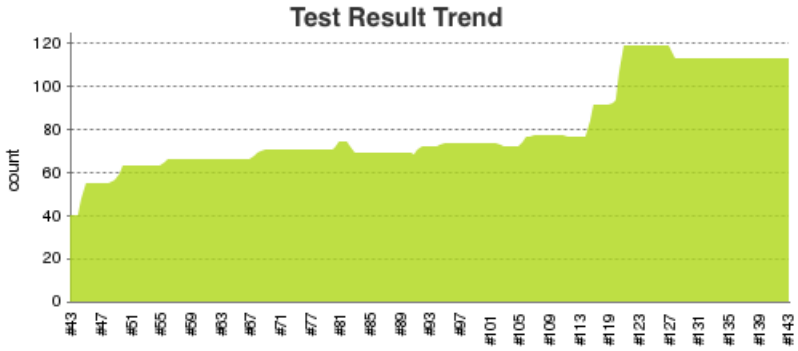


**Figure 4. Number of file writer automated unit tests against build number since November 2017.**

## 7.5 Code Quality

A code quality initiative was started after a decision at the Elettra integration milestone meeting. The motivation was to improve software maintainability, reliability, testability, extensibility and readability. As part of this initiative the following items have been actioned: the LLVM coding standard [17] has been adopted; the usage of automated analysis and test tools such as ClangFormat [20], Cppcheck [21] and Valgrind [22] has been extended; pull request templates have been created for merging changes to the source code repositories, with checklists and fields for description of the changes; and, group code reviews have been introduced. The code reviews are meetings in which a code base or change sets are reviewed by a group of developers, usually including DMSC and the partner institutes and using screen sharing and teleconferencing. The following group code reviews were carried out on data aggregation and streaming software:

**Table 1. List of code review sessions on data aggregation and streaming software, summarising the focus area and main resulting suggestions.**

| Date | Project | Focus | Suggestions from review |
|---|---|---|---|
| March 9, 2018 | NeXus File Writer | General look at code<br>• `CommandHandler`<br>• `FileWriterTask`<br>• `HDFFile`<br>• `HDFWriterModule`<br>• `Master`<br>• `json`<br>• `ev42_rw` | • Add comments in various places<br>• Remove dead code<br>• Avoid `using std::`<br>• Better naming for functions<br>• Use exceptions<br>• Separate classes into own files |
| April 6, 2018 | EPICS to Kafka Forwarder | General look at code<br>• `Kafka`<br>• `Main`<br>• `MainOpt`<br>• `Stream` | • Use pass-by-ref<br>• Add logging<br>• Better variable names<br>• Use `make_shared` rather than `reset`<br>• Use encapsulation<br>• Make single parameter constructor `explicit`<br>• Reduce number of classes per file<br>• Remove redundant code<br>• Use `const` where appropriate |
| April 12, 2018 | EPICS to Kafka Forwarder | Pull request to switch JSON library used | • Use functions for clarity<br>• Log unknown commands<br>• Keep lambdas short otherwise use function<br>• Use exceptions rather than return codes |
| April 26, 2018 | NeXus File Writer | Pull request to implement writing of fast sample environment data | • Make single parameter constructor `explicit`<br>• Add comments |

| | | | |
|---|---|---|---|
| | | | • Use asserts where appropriate |
| May 24, 2018 | EPICS to Kafka Forwarder | Pull request with further JSON abstraction | • Improve naming<br>• Add documentation/comments<br>• Use C++11 string literals<br>• Use `std::vector::at()` and `std::map::at()`<br>• Remove/rename unnecessary namespaces |
| June 15, 2018 | NeXus Streamer, EPICS to Kafka Forwarder and NeXus File Writer | Multiple pull requests | • Profiling in the NeXus Streamer<br>• Cppcheck warnings in the forwarder: decision required on which warnings to prioritise<br>• File writer: minor suggestions about comments and clarity |
| June 21, 2018 | EPICS to Kafka Forwarder | Multiple pull requests | • Prefer to use `make_shared` and `make_unique`<br>• Remove unused/unnecessary variables<br>• Unit tests names are *snake_case* |

## 7.6 Tests and Deployments

To verify that the data aggregation and streaming system satisfies its requirements, various types of tests have been carried out [3]. As mentioned in the previous sections, usage of automated testing tools in commit stage tests has been extended; the automated integration test has also been improved; the build system has been updated for better reproducibility; and, new tools have been adopted for dependency management. The following sections detail these activities.

### 7.6.1 Build Server

Jenkins [23] continues to be used as the build server for the data aggregation and streaming software, providing a central location for verifying build status and test results. To ensure builds happen in a well-defined, isolated environment, most of the Jenkins jobs now run inside Docker [24] containers which are created from a set of customised images and destroyed after builds complete. These container images include all of the required build and testing tools, including CMake [25], Conan [16], Cppcheck [21] and Valgrind [22].

Three representative operating systems have been selected as the base images for the build node containers, namely CentOS 7, Debian 9 and Ubuntu 18.04.

The current build server workflow is illustrated in Figure 5**Error! Reference source not found.**. All Jenkins jobs now use the newer Pipeline job type, in which the build commands are described in a file that is kept in each project's source code repository. A Jenkins plugin scans the repositories and automatically adds jobs when that file is found in the source code root. The plugin also manages branches and pull request builds. Software artefacts, including runtime dependencies, are archived upon build completion to ease deployment.

An internal Conan server is used to cache built dependencies, so they do not need to be rebuilt from source on every build. Private configuration files (e.g. some deployment information) are kept in private repositories in the DMSC GitLab server [26].

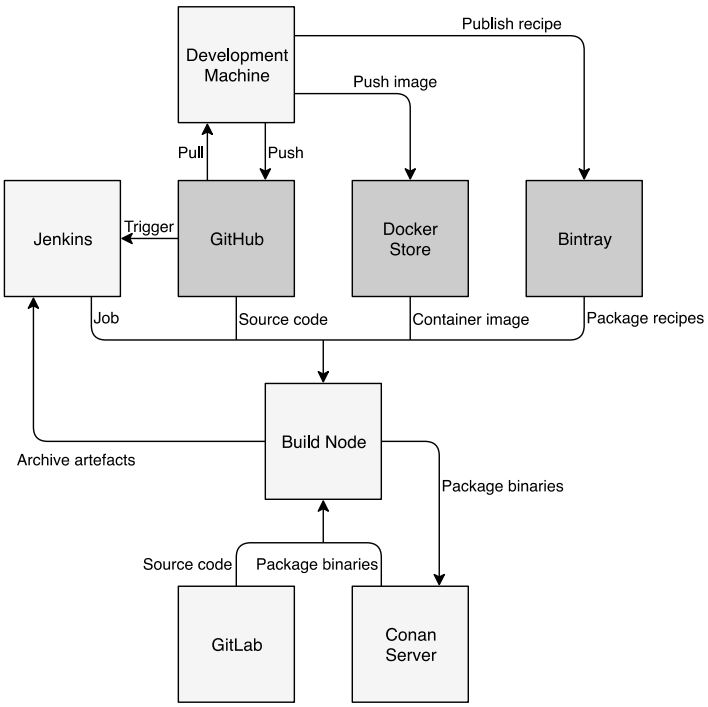


**Figure 5. Development workflow for automated build and testing on Jenkins, using Docker containers for reproducibility and isolation, and dependency management with Conan packages.**

### 7.6.2 Dependency Management and Deployment

Conan is a C/C++ package manager for software developers [16] and is now being used for managing software dependencies for the data aggregation and streaming software projects. A Conan package is based on a recipe, which is a Python file containing instructions to build a particular library or application, including a list of its required dependencies. Software using this package can download a binary version, if it is available, or alternatively obtain the recipe and build the binary package locally.

Bintray [27] provides free Conan package repositories for open source projects, and Conan includes a simple package server that can be run locally. A large number of different community-maintained, open source packages are available from Bintray, and recipes for packages required for the data aggregation and streaming projects (either missing or requiring customisation) have been created and uploaded to an ess-dmsc repository [28]. As there is limited bandwidth available for the open source repositories, binary versions of the ess-dmsc packages are kept in a local Conan server running in a virtual machine in the DMSC computing infrastructure. When a build is started, Jenkins first tries to obtain the dependency from the local Conan server, as having the pre-built binaries available greatly reduces the overall build

time. Packages are only required to be built from recipes if the appropriate binary is unavailable, for example, if a newer version is required. Community package binaries are also uploaded to the local server.

Conan provides integration with CMake, in the form of tools for generating CMake files for inclusion by projects, and also through the possibility of being invoked from a CMake file. The latter approach has been adopted by many of the data aggregation and streaming projects, as it allows build configuration and dependency download and installation to be carried out with a single command.

A set of Ansible [29] scripts contains the commands to configure the build and test environments, as well as to deploy data aggregation and streaming software and its dependencies. The runtime dependencies provided by Conan packages are included in the archived artefacts, making deployment simpler. These scripts and configuration files are kept under version control in a private GitLab source code repository.

### 7.6.3 Integration and Performance Testing

The integration test environment in the DMSC computing infrastructure has been updated and now more closely resembles a production-like environment. It used to run on three virtual machines that were shared by the multiple applications (e.g. Apache Kafka, EPICS to Kafka Forwarder and NeXus File Writer). This environment now comprises 13 separate virtual machines, with different services running on different servers (with the exception of Kafka and ZooKeeper, which share a machine).

Tests have been improved with the addition of checks and reports, including server logs and Kafka messages from the file writer command and status topics. These messages help finding the cause of problems when the test fails. NICOS, the chosen Experiment Control Program (ECP), has also been integrated with the test and is now used to start and stop NeXus file writing. The integration test environment includes a Graylog [30] server for logging, and Graphite [31] and Grafana [32] for storage and visualisation of metrics; Figure 6 shows a sample Grafana screen with test metrics.


**Figure 6. Integration test metrics visualisation in Grafana.**

Performance tests have been run at PSI and Elettra and results showing the scaling behaviour of Kafka and the NeXus File Writer are available at [4]. For the period after BrightnESS a tender process has been initiated for 10 additional servers for the Kafka cluster in the Utgård laboratory at ESS and to host additional services. These additional machines will be the local near full scale test bed for further development and test deployments until ESS instruments and a dedicated data centre become available.

### 7.6.4 Facility Deployments

The data aggregation and streaming system components have been deployed at different locations and tested under realistic production-like conditions, interfacing to real beamline hardware and operating neutron instruments. In December 2017, the system was deployed to the V20 beamline at the Helmholtz-Zentrum Berlin (HZB), where an ESS test chopper has been installed for testing. This deployment includes Apache Kafka, the EPICS to Kafka Forwarder, the NeXus File Writer, and Graphite/Grafana for collecting metrics. It was able to successfully acquire chopper data through EPICS. The experience provided valuable insights into problems and limitations of the system, which were used to define tasks for improving its components. Another important aspect of the V20 deployment is the NICOS installation for experiment control, as it allows the ECP to be tested with the aggregation and streaming infrastructure in a real beamline.

At ESS, the Utgård laboratory has been used to stage three demonstration sessions in which an Apache Kafka cluster has been used to stream detector data from the Event Formation Unit (EFU), with visualisation in Grafana and using the detector commissioning tool Daquiri [33]. The detector data comprised both pre-recorded experiment data and live data being streamed from a detector module.

The system has also been successfully deployed at the ISIS Neutron and Muon Facility, where it is currently being used by scientists on three beamlines. This deployment uses a modified version of the system adapted to their experiment control and data acquisition infrastructure, as described in [4].

# 8 Outlook and Conclusion

Future activities in the data aggregation and streaming system development will include integration and performance tests with a dedicated Kafka cluster at the ESS Utgård laboratory and further integration with real beamline sample environment equipment and detectors, as well as the experiment control program NICOS. At larger scale, stability tests will be carried out to test the system's resilience under different (network) failure modes to ensure it operates reliably and appropriate error handling is in place.

Having a comprehensive unit test suite addresses the risks associated with code refactoring, the addition of new functionality and defect fixing, as it allows checking that the behaviour obtained through interfaces is what is expected. Components that interface to the data aggregation and streaming system, like the Event Formation Unit, Fast Sample Environment and NICOS are part of current and future tests at Utgård. The Conan package manager has been adopted to provide project dependencies which decreases both the time to build the software and for deployment. Also important is code documentation, where appropriate comments and descriptions of architecture and intention make it easier for developers to get acquainted with a code base and be able to modify it.

This task of BrightnESS Work Package 5 produced results that provide confidence that the proposed data aggregation and streaming solution is viable and will be able to deliver the functionality, performance and availability required of it for the commissioning and operations of neutron instruments at ESS. As part of the code quality initiative, the number and coverage of unit tests have been increased and documentation was generated and updated. ESS instruments will use a data aggregation and streaming system based on the open source distributed streaming platform Apache Kafka, combined with the EPICS to Kafka Forwarder and NeXus File Writer applications.

# 9 List of Publications

- A. H. C. Mukai et al., "Architecture of the data aggregation and streaming system for the European Spallation Source neutron instrument suite," *Submitted to JINST* (Reference [4])

# 10 References


[1] "Forward EPICS to Kafka," [Online]. Available: https://doi.org/10.5281/zenodo.1298374.

[2] "Kafka to NeXus," [Online]. Available: https://doi.org/10.5281/zenodo.1298376.

[3] "BrightnESS Deliverable 5.3: Beta-version data aggregator software," 2017.

[4] A. H. C. Mukai et al., "Architecture of the data aggregation and streaming system for the European Spallation Source neutron instrument suite," Submitted to *J. Instrum.*, preprint available at https://arxiv.org/abs/1807.10388.

[5] "Apache Kafka," [Online]. Available: http://kafka.apache.org/.

[6] "BrightnESS Deliverable 5.2: Report processing choices for detector types," 2017.

[7] "BrightnESS Deliverable 5.4: Report field data acquisition," 2017.

[8] "Experimental Physics and Industrial Control System," [Online]. Available: https://epics.anl.gov.

[9] M. Könnecke et al., "The NeXus data format," *J. Appl. Crystallogr.,* vol. 48, 2015.

[10] O. Arnold et al., “Mantid–Data analysis and visualization package for neutron scattering and μSR experiments,” *Nucl. Instr. Meth. Phys. Res. A,* vol. 764, 2014.

[11] “NICOS,” [Online]. Available: http://www.nicos-controls.org/.

[12] “FlatBuffers,” [Online]. Available: https://github.com/google/flatbuffers.

[13] “Streaming Data Types,” [Online]. Available: https://doi.org/10.5281/zenodo.1298378.

[14] “JSON for Modern C++,” [Online]. Available: https://github.com/nlohmann/json.

[15] “CLI11,” [Online]. Available: https://github.com/CLIUtils/CLI11.

[16] “Conan,” [Online]. Available: https://conan.io.

[17] “LLVM Coding Standards,” [Online]. Available: http://llvm.org/docs/CodingStandards.html.

[18] D. Werder et al., “EPICS Data Streaming and HDF File Writing for ESS Benchmarked Using the Virtual AMOR Instrument,” in *16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’17)*, Barcelona, 2017.

[19] “h5cpp,” [Online]. Available: https://github.com/ess-dmsc/h5cpp.

[20] “ClangFormat,” [Online]. Available: https://clang.llvm.org/docs/ClangFormat.html.

[21] “Cppcheck,” [Online]. Available: http://cppcheck.sourceforge.net/.

[22] “Valgrind,” [Online]. Available: http://valgrind.org.

[23] “Jenkins,” [Online]. Available: https://jenkins.io.

[24] “Docker,” [Online]. Available: https://www.docker.com.

[25] “CMake,” [Online]. Available: https://cmake.org.

[26] “DMSC GitLab,” [Online]. Available: https://git.esss.dk/.

[27] “Bintray,” [Online]. Available: https://bintray.com.

[28] “ess-dmsc Bintray repository,” [Online]. Available: https://bintray.com/ess-dmsc/conan.

[29] “Ansible,” [Online]. Available: https://www.ansible.com.

[30] “Graylog,” [Online]. Available: https://www.graylog.org.

[31] “Graphite,” [Online]. Available: http://graphite.readthedocs.io/en/latest/.

[32] “Grafana,” [Online]. Available: https://grafana.com.

[33] “Daquiri,” [Online]. Available: https://github.com/ess-dmsc/daquiri.

[34] “PyEpics,” [Online]. Available: http://pyepics.github.io/pyepics/.

[35] “Confluence Kafka,” [Online]. Available: https://github.com/confluentinc/confluent-kafka-python.

# 11 Appendix A: A. H. C. Mukai et al., “Architecture of the data aggregation and streaming system for the European Spallation Source neutron instrument suite,” *Submitted to JINST*

Prepared for submission to JINST

# Architecture of the data aggregation and streaming system for the European Spallation Source neutron instrument suite

**A. H. C. Mukai,[a,1] M. J. Clarke,[a] M. J. Christensen,[a] J. M. C. Nilsson,[a] M. G. Shetty,[a] M. Brambilla,[b] D. Werder,[b] M. Könnecke,[b] J. Harper,[c] M. D. Jones,[d] F. A. Akeroyd,[c] C. Reis,[e] G. Kourousias[e] and T. S. Richter[a]**

[a] *European Spallation Source ERIC,*
*Ole Maaløes Vej 3, 2200 Copenhagen N, Denmark*
[b] *Paul Scherrer Institut,*
*5232 Villigen PSI, Switzerland*
[c] *ISIS Neutron and Muon Source, Science and Technology Facilities Council,*
*Rutherford Appleton Laboratory, Didcot, OX11 0QX, United Kingdom*
[d] *Tessella,*
*26 The Quadrant, Abingdon Science Park, Abingdon, Oxfordshire, OX14 3YS, United Kingdom*
[e] *Elettra Sincrotrone Trieste,*
*Strada Statale 14, km 163.5, 34149 Basovizza, Trieste, Italy*

*E-mail:* afonso.mukai@esss.se

Abstract: The European Spallation Source (ESS) will provide long neutron pulses for experiments on a suite of different instruments. Most of these will perform neutron data acquisition in event mode, i.e. each detected neutron will be characterised by one absolute timestamp and pixel identifier pair. Slow controls metadata from EPICS, such as sample environment measurements and motor positions, will also be timestamped at their source, so that all data and metadata are streamed as a list of events instead of histograms. A flexible data aggregation and streaming system is being developed combining both open source third-party software and in-house development. This is to be used at ESS and other neutron scattering facilities like ISIS and SINQ, replacing legacy solutions by a shared software collection maintained by a cross-facility effort. The architecture of the Apache Kafka-based system, its metadata forwarding and NeXus file writing components are presented, along with test results demonstrating their integration and the scalability in terms of performance.

Keywords: Data acquisition concepts, Computing (architecture, farms, GRID for recording, storage, archiving, and distribution of data), Software architectures (event data models, frameworks and databases)

[1] Corresponding author.

## Contents



## 1 Introduction

The European Spallation Source (ESS) is a spallation neutron source under construction in Lund, Sweden. A linear accelerator produces a high intensity proton beam which is guided on a tungsten target, with the spallation process generating fast neutrons from the heavy tungsten atoms. These fast neutrons are slowed down in a moderator and then guided onto samples with which they will interact. From the interaction of samples with neutrons, scientists can learn about the properties of materials. Applications include fundamental solid state physics, materials science, crystallography, biology and archaeology.

ESS will operate as a user facility: external users will come to perform neutron scattering experiments using the provided high brightness, long pulse neutron beam. A suite of instruments covering different techniques, such as imaging, small angle scattering, reflectometry, diffraction and spectroscopy, will be able to use the unprecedented flux there for experiments [1].

Due to the high brilliance neutron beam, high performance computing is needed for data processing. Located in Copenhagen, Denmark, the ESS Data Management and Software Centre (DMSC) develops, maintains and operates a software and hardware infrastructure for tasks ranging from data acquisition, aggregation and streaming to reduction and analysis, both in real time and offline. The DMSC works in close collaboration with in-kind and project partners across Europe,

including Paul Scherrer Institut (PSI) in Switzerland, the Science and Technology Facilities Council (STFC) in the United Kingdom, and Elettra in Italy.

Most ESS instruments will acquire neutron data from detectors in event mode, i.e. as a list of pixel number-timestamp pairs. Slow metadata obtained from devices using the Experimental Physics and Industrial Control System (EPICS) also come in the form of timestamped values [2]. An aggregation and streaming software architecture based on Apache Kafka [3] will make these data available to applications performing tasks such as file writing using the NeXus format [4] and live data reduction with feedback to the experiment control as well as visualisation.

Due to the high brightness of the ESS neutron beam, the solutions for data acquisition must be able to cope with correspondingly high data rates, making performance an important requirement, as will be discussed in this article. Considering the future possibilities of expansion of the ESS instrument suite and of upgrades to initial instruments, good scalability of the solution is essential. The selected open source components and the design of the in-house developed software take this into consideration.

In this article, the overall architecture of the data aggregation and streaming system is described, with a focus on the core components: Apache Kafka, the EPICS to Kafka Forwarder and the NeXus File Writer. Section 2 discusses data acquisition in event mode, the requirements arising from it and also the ESS approach to timing and timestamping. In section 3, the architecture and components are introduced and discussed. The current deployments of the system as well as integration and performance scalability tests are presented in section 4, with conclusions summarised in section 5.

## 2 Event mode acquisition

Acquisition of neutron instrument data is frequently done in histogram mode, in which detector data over a region is acquired for an interval and presented as a set of accumulated counts. In event mode acquisition, each neutron count on the detector generates an individual event consisting of a pixel number, which uniquely identifies the location where the neutron hit the detector, and a timestamp. Acquired data are sent forward as a sequence of events. Figure 1 illustrates the difference between these two modes of data acquisition.

Postponing histogram generation provides additional flexibility to the data acquisition process, as the conversion of neutron detection events into a histogram results in loss of information (the individual timestamps are lost and the final histogram only gives information as detailed as the binning and integration time allow). Not only can the data in an event list later be binned in any desired resolution, individual (sets of) events can also be filtered out depending on metadata parameters like, for example, sample environment information. In ESS, event mode acquisition will occur with no hardware veto, that is, data acquisition will not be automatically inhibited should a neutron chopper be out of phase. As chopper metadata will be recorded, filtering can be done by discarding events for intervals where choppers were out of phase, or by taking into account the instantaneous chopper phases, as well as source and target parameters and correcting the neutron wavelengths for it.

The data aggregation and streaming system will receive neutron data and metadata from different sources. Event data come from Event Formation Unit (EFU) software running on servers that interface with the detector readout electronics and obtain the pixel identifier-timestamp pair

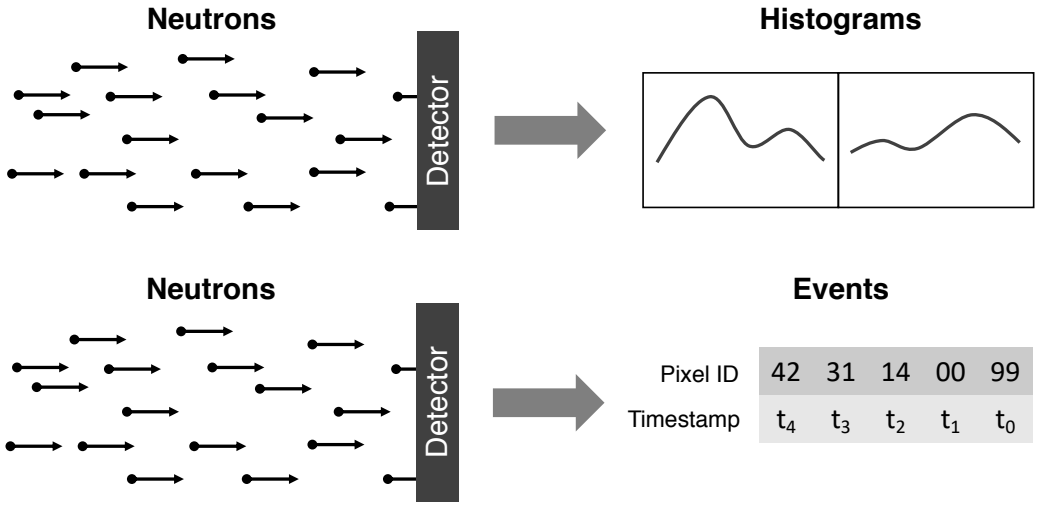


**Figure 1**. Data acquisition in histograms and event mode. In the first mode, counts from neutrons incident on the detector are accumulated and sent as a histogram. In the latter, pairs of timestamp and pixel identifier are sent for each detected neutron in a list of individual events.

from the raw detector data [5]. Figure 2 shows a screenshot of the detector commissioning tool Daquiri [6] with examples of event data before and after processing by the EFU. EPICS metadata will be sent by the EPICS to Kafka Forwarder application, which subscribes to value updates from multiple Input/Output Controllers (IOCs) that publish the readings for beamline devices like temperature controllers, motors and choppers. The highest data rates are expected to arise from neutron event data; these will be discussed next, followed by the ESS approach to timing and timestamping.

### 2.1 ESS instrument data rates

As a result of the high neutron beam brightness, the expected event data rates for ESS instruments are correspondingly high. Each detector pixel identifier and timestamp will be encoded in a pair of 32-bit numbers [7]; using these values, table 1 presents estimates of the data rates expected to be sent from the EFUs into the aggregation and streaming system [5].

**Table 1**. Anticipated neutron and detector event rate estimates for some early ESS instruments.

| **Instrument** | **Global Average Rate** [MHz] | **Data Rate** [MB/s] |
|---|---|---|
| C-SPEC | 10 | 80 |
| ESTIA | 500 | 4000 |
| FREIA | 100 | 800 |
| LoKI | 37 | 300 |
| NMX | 5 | 40 |
| SKADI | 37 | 300 |
| T-REX | 10 | 80 |

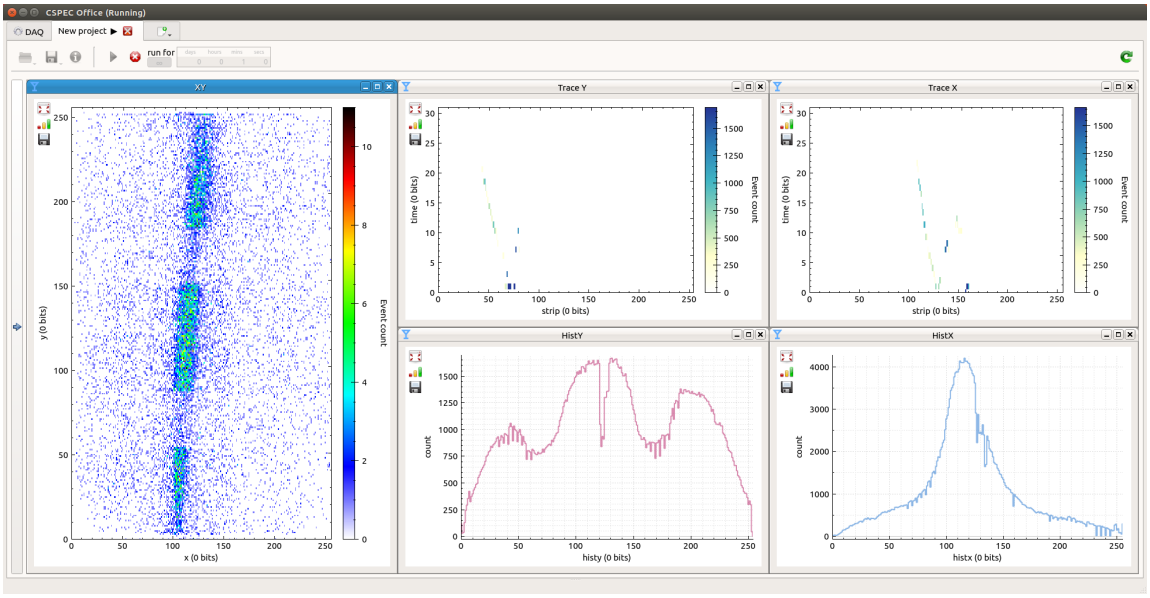


**Figure 2**. Screenshot of the detector commissioning tool Daquiri with detector prototype test data. The two smaller plots on the top show event traces (for a single event) before processing by the EFU. The plot on the left shows event counts on a detector surface after processing by the EFU.

As the table shows, expected rates cover a range of almost three orders of magnitude, from 10 MB/s to 4 GB/s. In addition to the bare detector neutron rate, the metadata from beam monitors, choppers, (fast) sample environments and so on also need to be considered. It is expected that the total metadata volume can be up to the order of the neutron data for an average instrument. This sets the requirements for the rates the system must support for online visualisation and file writing. The accelerator will be ramped up in power over a few years and instruments will be built and commissioned over a period of time, so ESS will not initially operate at maximum rates; the aggregation and streaming solution should therefore be scalable, allowing it to expand by adding more resources as necessary.

## 2.2 Timing system and timestamping of data

For devices that produce data or metadata, all readout values will be timestamped at the source (with deterministic latency) using information from the ESS timing system. In this system a tree topology of event receivers with an event generator at the root is used for distributing trigger and clock signals, timestamps, and accelerator parameters across the facility with deterministic and compensated latency. In normal operations, the ESS neutron pulses of 2.86 ms duration will have a repetition rate of 14 Hz. The timing system will make available beam parameters such as the pulse length and proton beam energy and will be integrated with EPICS [8].

Neutron detector data acquisition in event mode relies on information that is provided by the ESS timing system, such as accurate timestamps for neutron events coming from the detector readout electronics and accelerator pulse sequence numbers. Furthermore, to be able to make sense of data and process the events, sample conditions must be known at any given instant; for that, the timestamped metadata are used, using values provided by EPICS process s. As all the

timestamping happens at the data source, later transport over normal computer networks, software data aggregation and processing in standard non-realtime systems and so on do not distort the data by adding latency. Throughput times for the data must be kept reasonably low to allow live experiment data visualisation by the user and enable useful automatic feedback from live data processing to the control of the ongoing experiment.

## 3 Data aggregation and streaming

Data acquisition in event mode produces a stream of pixel identifier-timestamp pairs, which are subsequently processed by other applications and combined with metadata to enable a user to make sense of the experiment being performed. By selecting a system architecture based on aggregation and streaming [9], the data publishers do not have to be directly connected to every data subscriber, thus decoupling the two categories and allowing them to scale separately. A common serialisation format is needed to ensure the different applications all understand the data in the same way.

An overall view of the adopted system architecture is presented in figure 3. The data sources depicted in that diagram are the Event Formation Units; the Fast Sample Environment, which sends event data from fast readout systems such as alternating electric and magnetic fields; and the EPICS to Kafka Forwarder, which sends experiment metadata from EPICS IOCs. Subscribing to these data are the File Writer, which generates files conforming to the NeXus format [4] for offline reduction and analysis, and the data reduction application Mantid [10] performing live data reduction to enable online visualisation. The streams of data are brokered through an Apache Kafka cluster, which is also used for sending and receiving control and status messages. In the next sections the central components for aggregation and streaming are discussed in more detail.

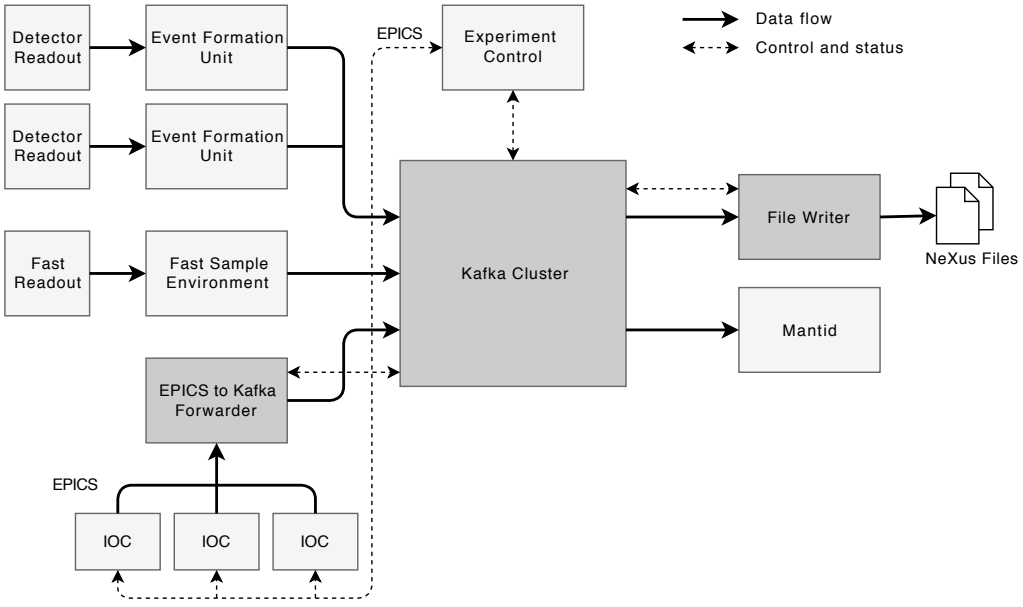


**Figure 3**. System architecture overview. Apache Kafka provides the central infrastructure that connects data producers and consumers. Components in dark grey are the main focus of the data aggregation and streaming and are described in this publication.

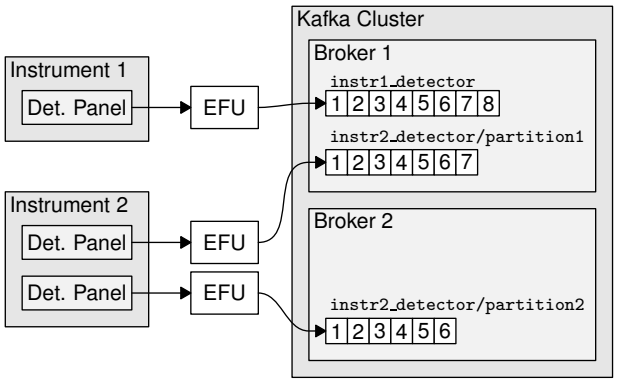


**Figure 4**. Neutron detector data from Event Formation Units sent to different Apache Kafka topics and partitions. Partitioning of topics across different Kafka brokers provides scalability to handle high data rates, as each broker runs on a server with dedicated network interfaces and storage resources.

The decoupling of the system components combined with the common serialisation format allows different facilities to plug in their own specific applications, while using the other parts of the infrastructure. An example of such use of the system by the ISIS Neutron and Muon Source is presented in section 4.3.

### 3.1 Apache Kafka

Apache Kafka is an open source distributed streaming platform that provides a scalable solution for applications to publish and subscribe to streams of data [3]. Kafka has been selected as the central technology for aggregation and streaming based on its rich set of implemented functionality, scalability, large and active user and developer communities, and the availability of good documentation [11].

In Kafka terminology, the *producers* (publishers) write data to named channels known as *topics*, to which *consumers* can subscribe. A cluster is formed by a number of Kafka brokers, with the possibility of partitioning topics across different brokers for scalability and replicating them for availability in case of broker failure. Data are persisted to disk for a configurable time or storage limit in the cluster, allowing consumers to obtain historical data. The cluster rate capacity can be increased by adding more brokers and partitioning topics across them.

Figure 4 illustrates how topic partitioning can be used to allow detector readout to scale by writing parts of the event streams to different Kafka brokers. It also shows that the intention is to use one common Kafka installation for all instruments. This shared system can cope with occasional peak loads from instruments far above their average rate, without needing to over supply capacity at every instrument. In addition to the depicted scaling out, Kafka will also be used to provide redundancy by duplication of the topics for the raw data storage, so experimental data is safely persisted early on.

Fast sample environment data will also be sent to the cluster using an EFU-based solution. For lower rate data sources, such as EPICS metadata, whose values are updated at rates in a typical range of 0.1 Hz to 100 Hz, the same Kafka topic can be used for multiple sources. The EPICS to

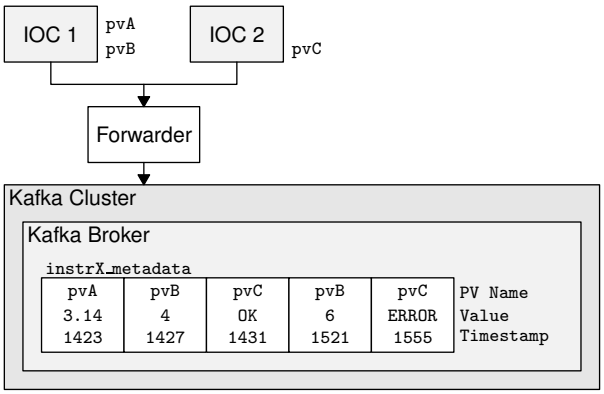


**Figure 5**. EPICS to Kafka Forwarder multiplexing data from different EPICS process variables (PVs) using the same Kafka topic.

Kafka Forwarder serialises and multiplexes different sources over a single topic, as illustrated in figure 5.

### 3.2 Serialisation and schemas

As the aggregation and streaming system is comprised of distributed producers and consumers that exchange data and commands through Apache Kafka, a well-defined common set of schemas is required to ensure the applications can make sense of the data being exchanged. Google FlatBuffers, an efficient cross-platform serialisation library [12] has been chosen for serialising neutron event data and beamline metadata. A common Streaming Data Types repository with an agreed procedure for introducing changes is used for controlling the schemas that define the message format [7]. Schemas can include other schema files, a feature that is used in the neutron event schema to add facility-specific information to them.

Serialised messages are not self-describing (i.e. they cannot be deserialised without knowledge of the schema), but include a four-byte file identifier; this is used to identify the schema required to deserialise the message by the consuming application. A convention of prefixing the schema file name with the identifier has been adopted to help identify the files in the repository. The FlatBuffers library allows schema fields to be optional or required, and fields can be added and deprecated. FlatBuffers provides a compiler that takes schemas as input and generates source files in a number of programming languages, including C++ and Python, the main languages used by the ESS aggregation and streaming programs. These source files provide language-specific methods for serialising and deserialising messages to and from buffers according to a given schema.

Apache Kafka is also used for the exchange of control and status messages between applications. Examples of control messages include commands for starting and stopping file writing by the Experiment Control Program, and both the EPICS to Kafka Forwarder and the NeXus File Writer send status messages reporting their state and activity periodically. These messages are currently sent in JSON format to dedicated Kafka topics, with a low rate of the order of 1 Hz.

### 3.3 EPICS Forwarder

The EPICS to Kafka Forwarder [13] is the component in the data aggregation and streaming architecture that monitors a list of EPICS process variables and makes these updates available for the rest of the components via the Kafka cluster.

Monitoring of the process variables is done using EPICS version 4, which supports both the older Channel Access protocol and the newer pvAccess [14]. The list of monitored variables can be set in a configuration file that is read by the EPICS to Kafka Forwarder at startup and can also be modified via JSON command messages, which are delivered through Kafka.

One or more conversion modules can be attached to each monitored process variable, which take the responsibility to convert the received EPICS value into a buffer for the desired schema, as described in section 3.2. It is also possible to attach an optional conversion module to several monitored process variables to cater for more elaborate setups. The converted update is delivered to the configured topic in the Kafka cluster (see section 3.1) via the C/C++ librdkafka library [15], which handles the network connections, buffering and delivery reports.

The EPICS to Kafka Forwarder features a highly parallel design based on worker queues and a configurable number of worker threads. It is designed to handle updates at a high frequency, which is achieved by keeping the EPICS update callback as lightweight as possible, while still providing the functionality to enable concurrency in the later stages of the forwarder pipeline. The update of the process variable can go through the pipeline stages while the thread that invokes the monitor callback, which is owned by the EPICS library, has already returned control to the library.

At runtime, the forwarder sends status and statistics about the monitored process variables to the Kafka cluster in JSON format. This information is useful for system monitoring by the Experiment Control Program and for diagnostics.

### 3.4 Writing data to file

While the Kafka cluster partitions persist the full stream of experimental data in roughly temporal order as they are measured, it is beneficial for the performance of offline data reduction or analysis algorithms to be able to access the data in larger homogeneous chunks. File-based storage is also well established in computing architectures and in the scientific community with agreed, self describing formats that can be a long term scientific record for experiments at ESS.

NeXus is a common data format for neutron, X-ray and muon science [4] and is based on the Hierarchical Data Format 5 (HDF5) [16]. Many institutions and existing pieces of software use NeXus. By opting to store data in this format the ESS benefits from existing support in software, including Mantid [10]. The NeXus File Writer [17] creates HDF5 files that contain a configured set of the available data and metadata from experiments.

HDF5 stores data in a format analogous to a filesystem. Directory-like objects called *groups* contain child groups and *datasets*. Datasets are analogous to files and contain a scalar value or a multidimensional array of data which may also be compressed; datasets and groups can each have *attributes* containing supporting metadata. Links can be used in place of groups or datasets and can link to locations in the same file or in external files.

The NeXus format defines a collection of classes, each of which has a set of required or optional child datasets, classes and attributes. HDF5 groups in a NeXus file declare their conformance to a

NeXus class with an attribute called `NX_class`, which has a value of the name of the class. Figure 6 depicts this for a simple hierarchy. Classes exist to support a wide range of different data [18], including images, histograms and time series data. They can also describe details about how the experiment was carried out, such as who carried out the experiment and when, information about the sample being studied, what components comprise the instrument, the instrument's geometry, and any other data required for the analysis of an experiment at a neutron source or synchrotron facility.

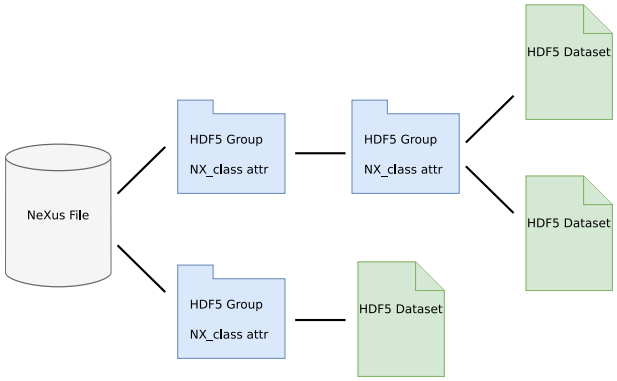


**Figure 6**. Diagram showing the hierarchy of a NeXus file, which comprises HDF5 datasets and groups that conform to NeXus class definitions.

As mentioned in Section 2, key data types for the ESS are neutron detection events and time series data from choppers or measurements of the sample environment. These types of data are well supported in NeXus by the `NXevent_data` and `NXlog` classes, respectively. Each of these classes contain datasets of values and corresponding timestamps. These *time-value* datasets are supplemented by *cue* datasets which record the index in the value and timestamp datasets at particular times. If only a subset of the time range recorded in the value-timestamp datasets is required to be read from file, this can be achieved by reading the smaller cue datasets and using the indices into the large datasets. This is therefore possible without having to read the entire time-value datasets, which may be slow or impossible in the case that they exceed available memory. This system is also particularly well suited to streamed data, as neutron detection events are batched in messages and a cue can be recorded for each message.

The file writer writes NeXus files upon requests received in the form of JSON commands via a Kafka topic. The request contains the structure of the NeXus file which is to be written. This description of the NeXus structure can contain the HDF5 group hierarchy, attributes, datasets which are already known at the time when the command is sent, and placeholders for datasets which are to be written as the corresponding data messages appear on configured Kafka topics. Because the file writer has access to historic data through the Kafka cluster, the commands can contain timestamps for the time range of the data to be written, which can be in the past or in the future. If start and stop times are not defined, the writing of a file starts as soon as the command is received by the NeXus File Writer and it stops on a corresponding stop command.

The data streams, which are specified in form of the NeXus structure in the JSON command, are handled by specified HDF5 writer modules. They process the messages that arrive on the

configured Kafka topic and are responsible for any additional processing and the actual writing of that data to the HDF5 file. This modular design allows easily extending the file writer to cater for new FlatBuffers schemas, for more specialised output formats or optimised ways of writing data to file.

## 4 Tests and results

To verify and demonstrate the viability of the proposed approach to data aggregation and streaming, a series of tests have been performed, covering the integration of the components and their performance. Some of the tests are automated and run as new versions of the applications are created, in a local computing infrastructure with virtual machines at the DMSC; others are run on demand on dedicated hardware. Furthermore, the system has been deployed to operating neutron facilities for evaluation. In the following sections, the different types of tests and their results are presented, along with a discussion of the deployments of the system to operating facilities.

### 4.1 Integration tests

An automated integration testing regime including the data aggregation and streaming software is in place in the software build and test infrastructure at DMSC, consisting of a Jenkins continuous integration server using the configuration management and orchestration tool Ansible. A test is triggered every time changes are made to the code bases of the Event Formation Unit, EPICS to Kafka Forwarder and NeXus File Writer. Raw data from detector prototype measurements are streamed to the EFUs, which process the signals into events and forward them to the Kafka cluster. A simulation EPICS server is used to generate process variable data for the EPICS forwarder, and the file writer is started and configured to write detector data and EPICS metadata to a NeXus file. Figure 7 shows the current setup; results are made available through the Jenkins server web interface, while metrics and logs can be visualised with Grafana and Graylog, respectively.

The integration test runs checks to verify the correct number of events have been sent from the EFUs. Metrics and log data are stored for the test runs and provide benchmark information such as Kafka data rates. A script verifies the data written to the NeXus file.

At the moment three dedicated servers are available at an ESS integration laboratory, where other beamline equipment such as motion control and sample environment devices are also located for testing purposes, along with detector prototypes. These servers are operated for manual tests and have been used for a successful demonstration of streaming event data from EFUs into Kafka with live visualisation. Work is in progress to integrate the Experiment Control Program and Fast Sample Environment application with the devices in the laboratory, and this will allow those components to be tested with the data aggregation and streaming system.

### 4.2 Performance tests

The performance of the aggregation and streaming system, including the NeXus File Writer, was investigated at PSI using four Intel Xeon E5-2697 v4 2.60 GHz servers, equipped with 252 GB of RAM and a GPFS file system connected via 4x Infiniband FDR. This system is limited by the fact that all the nodes share the same I/O interface. Furthermore, the same interface is used for communications to achieve optimal communication speed during the tests.

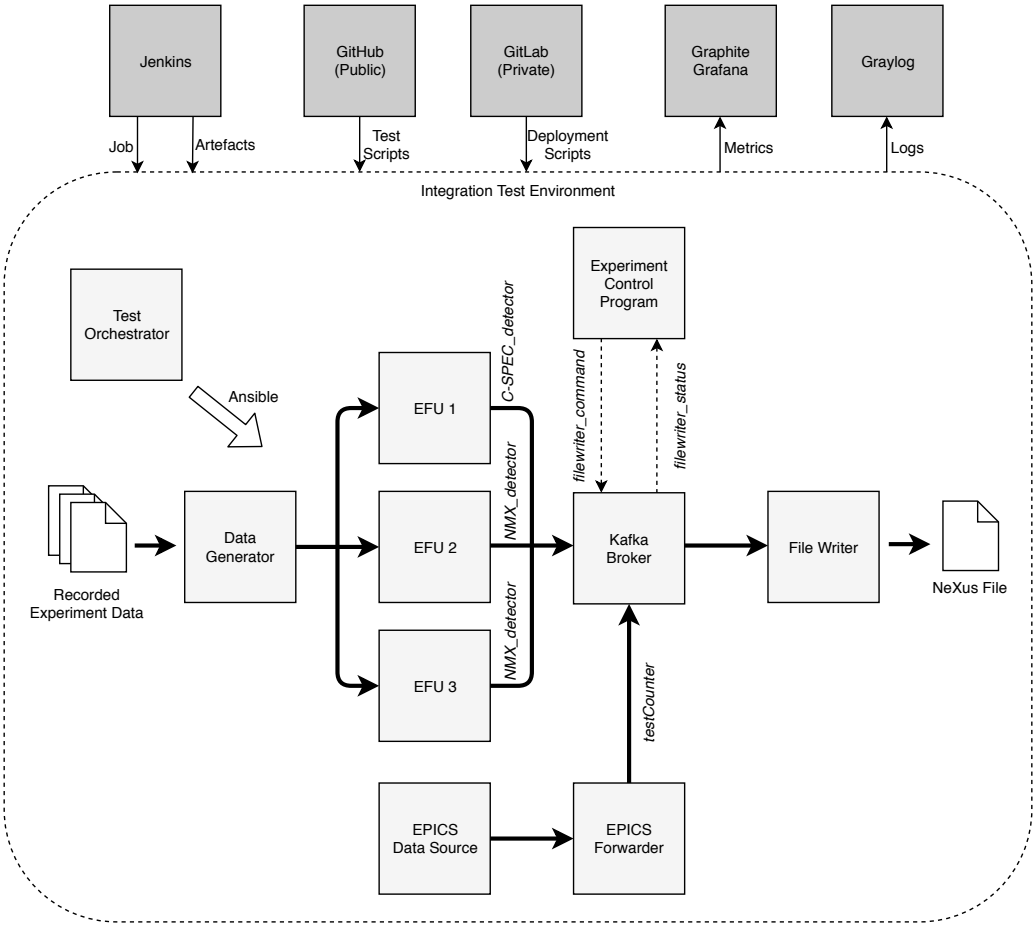


**Figure 7**. Integration test setup. The Jenkins build server triggers the test, which runs in virtual machines in the DMSC computing infrastructure (depicted in light grey). The test orchestrator node uses Ansible to connect to the other virtual machines and run the required services and commands. Data flow is represented by bold lines, status and commands by dashed lines, and topic names are written in italics.

Tests were performed using a virtual implementation of the PSI AMOR [19] instrument as data source. The output of a real measurement is converted into a neutron event stream and sent to a Kafka cluster; the number of events recorded in a single message can be varied in order to test the dependence of the system on the message size. A different number of brokers and partitions were used. Figure 8 shows the producer performance achieved using four Kafka brokers as a function of the message size and the number of partitions. The figure demonstrates that the overhead of using many small messages degrades the performance; this can be acceptable for infrequent metadata updates, but the bulk of the neutron data needs to be sent in larger packets, as done by the EFUs, to reach the required throughput. This system scales well with the number of producers when using different topics and the aggregate throughput of two producers is roughly double what is shown in the figure.

Figure 9 shows the results of scalability tests with a producer, the Kafka cluster and a file writer. For a different number of servers, the total throughput was measured varying the number of brokers, using one partition per broker and a 1 MB message size. The file writer runs on a separate

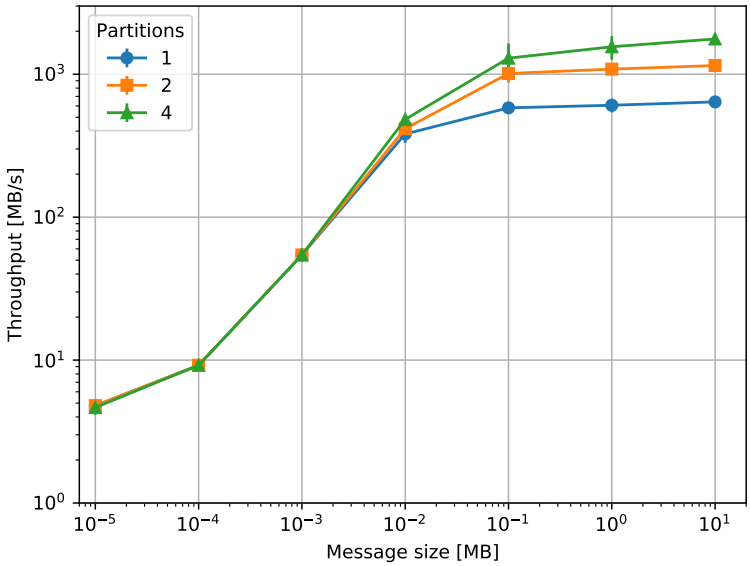


**Figure 8**. Kafka throughput as a function of the message size and the number of partitions. Two servers run the brokers, while a third server runs the event producer.

server, but in the setup with three broker machines, the producer shares the server with (at least) one Kafka broker. This may have a negative effect on the performance on the three server case (green line). Otherwise the data suggests a very linear scaling of the achievable throughput with the number of deployed servers or broker processes and in turns verifies the arguments made in 3.1 for choosing Kafka: that it enables an incremental growth of the infrastructure with raising need of the scientific instruments as ESS builds up, always keeping up with the requirements.

The performance of the HDF5 writing component is essentially limited by the available I/O. To be able to write HDF5 files at the expected high data rates, the file system layer has to be backed by an appropriately fast I/O layer. Furthermore, the File Writer exposes some parameters for fine tuning in order to achieve the best performance. A single process approach sustains about 1.2 GB/s, while a MPI implementation can reach about 4.8 GB/s; a broad discussion on the writing performances can be found in [20], including the effects of chunking, message size and buffering.

### 4.3 Deployments at operating facilities

The data aggregation and streaming system was deployed in December 2017 to the V20 beamline at the Helmholtz-Zentrum Berlin's BER II research reactor. That deployment comprises a Kafka broker, the EPICS to Kafka Forwarder and the NeXus File Writer, along with applications for obtaining system metrics. Running the software infrastructure there allowed debugging the software and its configuration. An ESS mini-chopper was installed in the beamline and the system was able to obtain its data via EPICS and write them to file.

Another current deployment of the system is running at the STFC's ISIS Neutron and Muon Source. Three instruments are using the EPICS to Kafka Forwarder and a Kafka cluster. The system

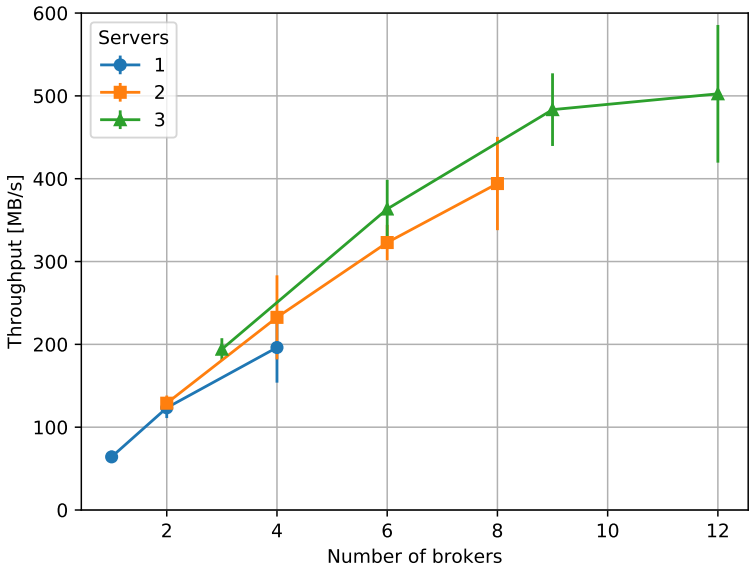


**Figure 9**. Scalability measurements: throughput as a function of the number of Kafka brokers with an event producer and a file writer, using one partition per broker.

architecture differs from the ESS architecture in that the detector data do not come from EFUs, but from data acquisition electronics through the Instrument Control Program (ICP), together with metadata; moreover, the ICP is responsible for writing NeXus files. The EPICS forwarder is used to send beamline metadata to Kafka, while the Mantid Live Listener consumes live data from the cluster. Grafana is also being used for metrics visualisation. In two of the instruments, the system is being run in parallel with the current facility system, but in one of them, which is a new instrument, the Kafka-enabled ICP is the main control program. The ISIS Kafka cluster runs on three servers with Xeon R5-2620 2.10 GHz 8-core processors, 32 GB of RAM and three 1.2 TB disks in RAID 0 (for each server).

## 5 Conclusion and future work

ESS will use a system architecture based on data aggregation and streaming for acquisition of neutron data in event mode and EPICS metadata, all of which will be timestamped at their source using the ESS timing system. An Apache Kafka cluster connects the applications producing and consuming these data, including Event Formation Units, Fast Sample Environment devices, EPICS Input/Output Controllers using the EPICS to Kafka Forwarder, Mantid and the NeXus File Writer. The solution can be tuned for balancing the throughput, latency and storage concerns arising from requirements, and also provides scalability and availability through partitioning and replication.

The Google FlatBuffers library is used for data serialisation and, with the schema definitions in a common code repository under source control, decouples the producer and consumer applications. This decoupling means parts of the system can be substituted by alternative components that use the appropriate schemas, making the system more generic and also usable at other neutron facilities.

Integration and performance tests have been run and demonstrate the viability of the solution for ESS, showing the desired functionality and scalability behaviour. Deployments to operating neutron facilities, like the Helmholtz-Zentrum Berlin's BER II research reactor and the ISIS Neutron and Muon Source, corroborate this.

In the future the integration of the forwarder and file writer applications with the Experiment Control Program will be improved and the whole system tested for robustness in a number of failure scenarios. End-to-end tests will be performed at the integration laboratory at ESS, using real beamline equipment and detector readout systems, as well as at the V20 beamline at HZB.

## Acknowledgments

This work is partially funded by the European Union Framework Programme for Research and Innovation Horizon 2020, under grant agreement 676548.

## References

[1] S. Peggs et al., *ESS Technical Design Report*, ESS-doc-274. Available at https://europeanspallationsource.se/sites/default/files/downloads/2017/09/TDR_online_ver_all.pdf

[2] *Experimental Physics and Industrial Control System website*, https://epics.anl.gov

[3] *Apache Kafka website*, http://kafka.apache.org/

[4] M. Könnecke et al., *The NeXus data format*, *J. Appl. Crystallogr.* **48** (2015). https://doi.org/10.1107/S1600576714027575

[5] M. J. Christensen et al., *Software-based data acquisition and processing for neutron detectors at European Spallation Source — early experience from four detector designs*, submitted to *J. Instrum.* Available at https://arxiv.org/abs/1807.03980

[6] *Daquiri*, https://doi.org/10.5281/zenodo.1298511

[7] *Streaming Data Types*, https://doi.org/10.5281/zenodo.1298378

[8] J. Cereijo García, T. Korhonen, J. H. Lee, *Timing System at ESS*, in *16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017. https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA088

[9] A. H. C. Mukai et al., *Status of the Development of the Experiment Data Acquisition Pipeline for the European Spallation Source*, in *16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017. https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA177

[10] O. Arnold et al., *Mantid–Data analysis and visualization package for neutron scattering and μSR experiments*, *Nucl. Instr. Meth. Phys. Res. A* **764** (2014). https://doi.org/10.1016/j.nima.2014.07.029

[11] A. Mukai, T. Richter, *Design report data aggregator software*, BrightnESS Deliverable Rep. 5.1. https://doi.org/10.17199/BRIGHTNESS.D5.1

[12] *FlatBuffers website*, http://google.github.io/flatbuffers/

[13] *Forward EPICS to Kafka*, https://doi.org/10.5281/zenodo.1298374

[14] *EPICS V4 website*, http://epics-pvdata.sourceforge.net/index.html

[15] *librdkafka code repository*, https://github.com/edenhill/librdkafka

[16] The HDF Group, *Hierarchical Data Format, version 5*, http://www.hdfgroup.org/HDF5/, 1997–2018.

[17] *Kafka to NeXus*, https://doi.org/10.5281/zenodo.1298376

[18] NeXus Format, *Base Class Definitions*, http://download.nexusformat.org/doc/html/classes/base_classes/index.html, 1996–2018.

[19] J. Stahn, A. Glavic, *Focusing neutron reflectometry: Implementation and experience on the TOF-reflectometer Amor*, *Nucl. Instrum. Meth. Res. A* **821** (2016). https://doi.org/10.1016/j.nima.2016.03.007

[20] D. Werder et al., *EPICS Data Streaming and HDF File Writing for ESS Benchmarked Using the Virtual AMOR Instrument*, in *16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17)*, Barcelona, Spain, Oct. 2017. https://doi.org/10.18429/JACoW-ICALEPCS2017-THPHA167