BrightnESS

Building a research infrastructure and synergies for highest scientific impact on ESS

H2020-INFRADEV-1-2015-1

Grant Agreement Number: 676548

Deliverable Report: D5.3 Beta-version 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.: 5.3	
	Deliverable Type: Report	
	Dissemination Level: Public	Contractual Delivery Date: 31.07.2017
		Actual Delivery Date: 18.08.2017
	EC Project Officer: Anna-Maria Johansson / Maria Vasile	

2 Document Control Sheet

Document	Title: BrightnESS_Deliverable_5.3	
	Version: 1.0	
	Available at: https://brightness.esss.se	
	Files: 1	
Authorship	Written by	Afonso Mukai
	Contributors	Carlos Reis, George Kourousias, Dominik Werder, Jonas Nilsson
	Reviewed by	Tobias Richter (WP5 leader), Matthew Jones, Roy Pennings (WP1 leader)
	Approved by	BrightnESS Steering Board

3 List of Abbreviations

API	Application Programming Interface
DMSC	Data Management and Software Centre (ESS)
ECP	Experiment Control Program
EFU	Event Formation Unit
EPICS	Experimental Physics and Industrial Control System
ESS	European Spallation Source
ESSIIP	ESS Instrument Integration Project
GELF	Graylog Extended Log Format
HZB	Helmholtz-Zentrum Berlin (Germany)
ICS	Integrated Control Systems (ESS)
IOC	Input/Output Controller
IP	Internet Protocol
JSON	JavaScript Object Notation
MLZ	Heinz Maier-Leibnitz Zentrum (Germany)
NTP	Network Time Protocol
PSI	Paul Scherrer Institut (Switzerland)
PV	Process Variable
REST	Representational State Transfer
STFC	Science and Technology Facilities Council (UK)
TCP	Transmission Control Protocol
WP	Work Package

4 List of Figures

Table of Contents

5 Executive Summary

This document concerns BrightnESS Deliverable 5.3: “Beta-version data aggregator software”. It marks the delivery of a working software prototype of the main component of the data streaming infrastructure, that is needed to determine the roadmap to the full completion of the task (D5.5 due July 2018). The development of the data aggregation software system is progressing ahead of the WP schedule. With the choice of the open source third party project Apache Kafka as the central component for aggregation and streaming, the WP5 participants could focus on the domain specific software components to retrieve and aggregate the data, and to write them to file. This maximises the impact of work package 5 for ESS and other neutron facilities as it provides better functionality, flexibility and fault tolerance of the data acquisition chain. In current integration tests, neutron event data - together with simulated instrument metadata - automatically traverse the readout pipeline from event formation to file writing through Kafka.

The full system is currently being tested and is iteratively being integrated with components developed outside of the WP, like the Experiment Control Program, Mantid and the Data Curation System. Tests on dedicated high-performance hardware are planned before the final

software delivery to evaluate the system performance and scalability under realistic conditions (using the test beamline at the Helmholtz-Zentrum Berlin) and establish hardware requirements for its deployment.

Services currently under development in WP5 include the EPICS Forwarder, which aggregates instrument metadata, and the NeXus File Writer, which subscribes to aggregated data and writes them to file. The Event Formation Unit, part of the BrightnESS Task 5.1, has also incorporated functionality to send neutron event data to Kafka, and is thus being integrated with the aggregation and streaming system. Common formats for data exchange among the applications are defined in a FlatBuffers schemas repository. All the components of the system are being regularly tested in an automated integration setup, which ensures applications can successfully send and receive data and interpret them according to the common schemas.

6 Report on Implementation Process and Status of Deliverable

ESS is a spallation neutron source currently being built in Lund, Sweden, and will operate as a user facility offering a high brightness neutron beam, in long pulses that can be tailored for adjusting resolution and bandwidth. Individuals or groups will submit proposals for performing experiments at one or more ESS instruments; these proposals will be competitively judged by peer review in order for users to be awarded time on an instrument. Neutrons provide a means to probe the structure and dynamics of atoms and molecules, have a high penetration power that allows studying bulk materials and can be used to probe properties of magnetic materials, among other applications. ESS instruments will enable studies in areas including life science, soft condensed matter, engineering materials and geosciences, and archaeology and heritage conservation.

As a consequence of the high brightness beam, experiments at ESS will generate large quantities of data. Neutron events at detectors will be associated with the sample and instrument conditions at interaction time and be stored in files for analysis. The data will also be employed for live reduction and visualisation, providing real-time feedback while the experiment is running. The Data Management and Software Centre (DMSC) is responsible for the acquisition and analysis of the scientific data from the ESS neutron instruments. The Data Aggregation Software being developed at the DMSC will aggregate neutron event data and metadata originating from sources such as sample environment, motion control and choppers; aggregated data will be consumed by software systems performing tasks including file writing and live data reduction and visualisation.

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 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.

The deliverable is the beta version of the data aggregation software system, which is publicly available. This document discusses the system architecture, its components and their integration, as well as external components with which interaction is required. Most of the components have been developed from scratch as part of the WP5 efforts. We present the types of tests being performed to ensure software quality for individual applications and for integration, lessons learned from these activities and plans for future steps.

7 Technical Content

This deliverable marks the completion of a working software prototype of the main component of the data streaming infrastructure, that is needed to determine the roadmap to the full completion of the task (D5.5 due July 2018).

7.1 General Overview

A typical neutron instrument contains several components for performing scientific experiments, allowing a user to position a sample, change its conditions and the properties of the neutron beam used to analyse it, and also acquire data, recording the results of the beam interacting with the sample. The data aggregation and streaming software system is responsible for collecting all the data and metadata from these sources and providing them via a single channel to systems which provides data analysis, experiment control and long-term storage.

The ESS instruments will generate large volumes of data, due to the high accelerator power and to neutron data acquisition in event mode. Neutron events and metadata will be generated by multiple components and provided using different protocols. The data will be consumed by applications such as Mantid, which performs live data reduction, and the NeXus File Writer, persisting experiment data to files. Instrument data aggregation allows these consumers to obtain the data from a single location, using a single protocol.

The data aggregation and streaming software system is being developed as part of BrightnESS at DMSC and by ESS in-kind partners. The software is at the state of a working demonstrator. It comprises a set of applications under development that are continually undergoing integration tests with each other as a part of the development cycle. The software (and the data acquisition system in general) needs to interface to different external components, including the Experiment Control Program (ECP), configuration databases, the Data Curation System and the User Office. Work on the integration and this activity has begun and will continue in the next step of the project.

In the following sections, we will describe these topics and the corresponding results achieved inside this WP in more detail, starting with the software components that are part of the data aggregation architecture, and external components. We will discuss the existing and planned tests that verify and demonstrate that the system requirements are being met, discuss problems and risks, and conclude with a roadmap.

7.2 Elements of the Pipeline

The entire data streaming pipeline consists of multiple components. The central part is Apache Kafka [1], a third-party open source project. With this exception all other services that build the backbone of the data aggregation and streaming system are initiated and developed as part of WP5. All source code is available from the ess-dmsc GitHub organisation repositories [2]. Instructions are provided in the repositories for building and running the software, and Jenkins [3] is used to automate builds and tests.

Figure 1 shows the overall system architecture with the Apache Kafka cluster in the centre. Data aggregation and streaming components such as the EPICS Forwarder, the Fast Sample Environment and multiple Event Formation Units obtain data from the sources on the instrument and send them to Kafka, as represented with the bold arrows on the *Kafka Producers* side. The bold arrows on the *Kafka Consumers* side represent the flow of data to consumers as Mantid and the File Writer. Dashed arrows represent the flow of metrics and log messages to auxiliary applications used for monitoring and debugging the system.

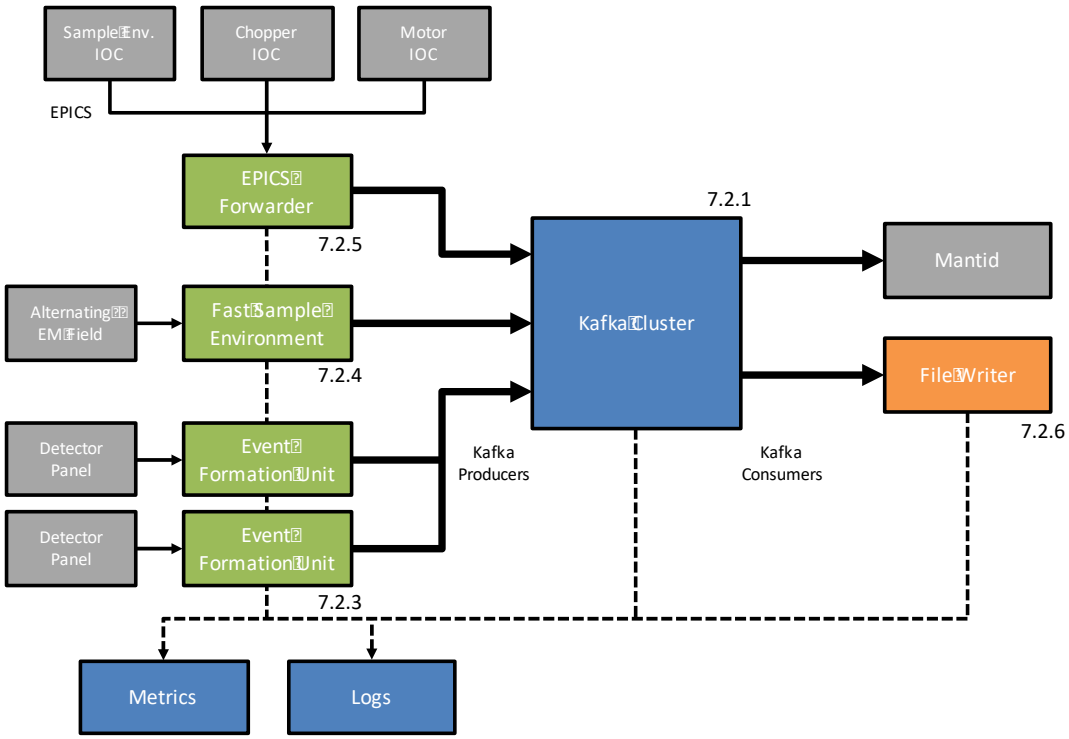

Figure 1. System architecture with component section numbers in report.

7.2.1 Apache Kafka

Apache Kafka is an open source software project for distributed data streaming. Multiple Kafka *brokers* form a *cluster*, which supports a publish-subscribe message queue pattern with configurable data persistence. A description of Kafka is given in the BrightnESS deliverable 5.1: *Design report data aggregator software* [4], which presents the criteria used for the technology decision. We only summarise Kafka's most important features here.

In Kafka, *producers* send data to a *topic* in a cluster. A *consumer* subscribes to a topic to receive messages, either from the instant the subscription starts or from a previous offset, provided the requested data is still available in storage on the cluster, given a retention policy. Consumers may also be grouped to distribute the processing load among different processes. Both producers and consumers can be developed using open source Kafka client libraries, for example librdkafka [5] for use with the C/C++. Figure 2 illustrates the relation between the Kafka cluster, producers and consumers.

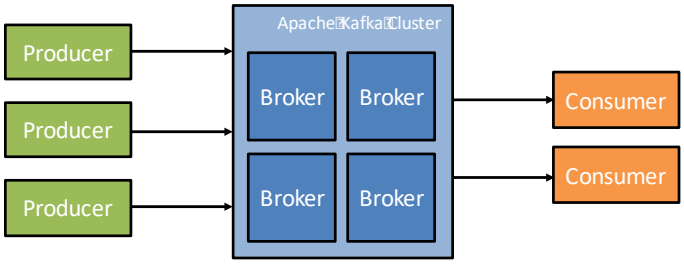

Figure 2. Apache Kafka cluster with producers and consumers.

Topics are named channels for messages that can be partitioned and distributed across brokers for scalability, and may optionally be replicated on different brokers for fault-tolerance. Data passing through Kafka is persisted to disk and kept for a configurable interval or amount of storage usage. Figure 3 shows the structure of a Kafka topic with two partitions; each

partition is on a different broker and writes are divided between them, thereby distributing the writing load. For ESS the retention policy has to guarantee that the File Writer has enough time to write data to NeXus [6] files, taking into consideration possible failures and delays.

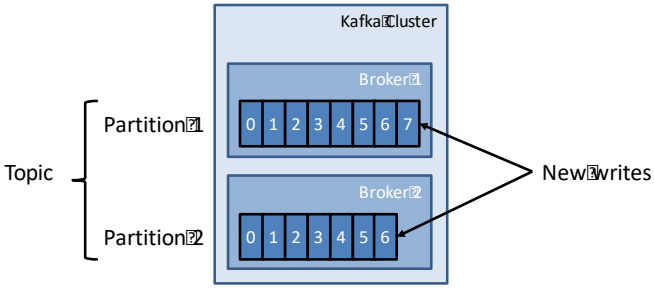

Figure 3. Kafka topic with two partitions.

While Kafka is was a functional third party project from the start of BrightnESS, significant effort was spent in the task to configure it and make the best use of its functionality for our neutron data streaming application. This will be covered to some extend in the section about the integration tests.

7.2.2 Conventions and Streaming Data Types

As part of WP5 the network data formats needed for communication between producers and consumers have been described in the FlatBuffers [7] serialisation scheme. FlatBuffers is an efficient, cross-platform format, chosen as the foundation of the format of the Kafka messages due to its low processing overhead to write and read data. Especially the numeric representations are the same as the in-memory representation on the machines used at ESS. FlatBuffers can represent data structures from simple pre-defined static structures to arbitrary recursive structures, and it allows for explicit differentiation between numeric data types, which makes it easy to interface with EPICS. More details about FlatBuffers are also available in the BrightnESS deliverable 5.1 [4].

The content of a FlatBuffers buffer is defined by a schema. Each schema is identified by a 32-bit identifier, which is contained in each transmitted FlatBuffers. By inspection of the FlatBuffers data packet, a receiver can therefore determine the correct schema for unpacking. This requires all schemas to be commonly available. It is possible to make backward compatible modifications to a schema which is already in use and a policy to that effect is in place. All schemas are available in the ess-dmsc streaming-data-types repository [8]. As shown in Figure 4, they are processed by the compiler to generate programming language-specific files, which are in turn included by software that produces or consumes messages.

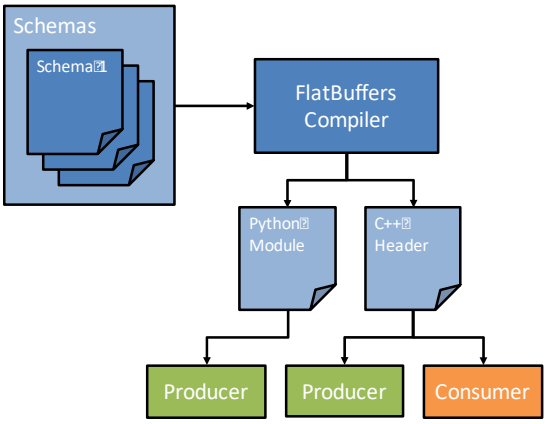

Figure 4. FlatBuffers schemas and files for language-specific includes.

The included files provide functionality for the application to convert data from an input format (e.g. detector events, EPICS metadata) to the agreed upon common serialised format. These data are then sent in messages to topics in the Kafka cluster. Consumers subscribe to certain topics and de-serialise the data into the appropriate format before using them. This process is illustrated in Figure 5.

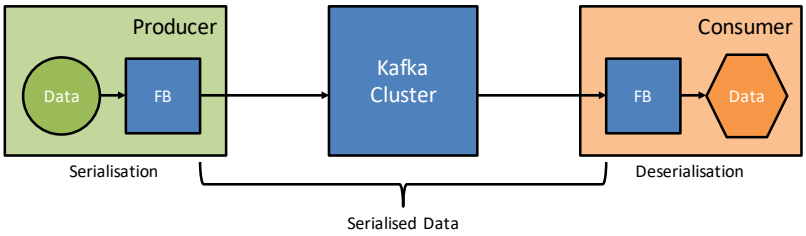

Figure 5. FlatBuffers messages serialisation, transmission and deserialisation.

A convention of using the FlatBuffers 4-character file identifier in the schema name has been introduced. Schema writers can pick an unused identifier by inspecting the existing files in the streaming-data-types repository, and the convention establishes that the file name must be prefixed by the identifier. A naming convention has been adopted to identify the Kafka topics corresponding to each data stream, in the format *<beamline>_<datatype>*, e.g. *HEIMDAL_monitors* and *C-SPEC_choppers*.

Not only aggregated data can be sent over Kafka topics, but also messages for controlling applications. This allows for asynchronous communication between applications, resulting in less coupling among them. This design is currently being evaluated.

Table I lists the FlatBuffers schemas currently used by the data aggregation and streaming applications, their producers and consumers. The sections below present these schemas.

Table I. FlatBuffers schemas, producers and consumers.

Schema	Description	Producers	Consumers
ev42_events.fbs	Neutron detection event data	Event Formation Unit	File Writer, Mantid
f141_epics_nt.fbs	EPICS version 4 Normative Types	EPICS Forwarder	File Writer
f142_logdata.fbs	Scalar and array data	EPICS Forwarder, Fast Sample Environment (potentially)	File Writer

ev42_events.fbs

```
// Schema for neutron detection event data

include "is84_isis_events.fbs"; // Include facility-specific definitions

file_identifier "ev42";

union FacilityData { ISISData } // SINQData, ESSData etc

table EventMessage {
  source_name : string;   // optional field identifying the producer type,
                          // for example detector type
  message_id : ulong;     // consecutive numbers, to detect missing or
                          // unordered messages
  pulse_time : ulong;     // time of source pulse associated with these
                          // events, nanoseconds since Unix epoch (1 Jan 1970)
  time_of_flight : [uint]; // nanoseconds measured from pulse time
  detector_id : [uint];   // detector on which the event was recorded
  facility_specific_data : FacilityData;  // optional field
}
root_type EventMessage;
```

f141_epics_nt.fbs

```
file_identifier "f141";

namespace BrightnESS.FlatBufs.f141_epics_nt;

struct timeStamp_t {
  secondsPastEpoch: ulong;
  nanoseconds: int;
}

table NTScalarByte   { value:  byte;   }
table NTScalarUByte  { value: ubyte;   }
table NTScalarShort  { value:  short;  }
table NTScalarUShort { value: ushort;  }
table NTScalarInt    { value:  int;    }
table NTScalarUInt   { value: uint;    }
table NTScalarLong   { value:  long;   }
table NTScalarULong  { value: ulong;   }
table NTScalarFloat  { value:  float;  }
table NTScalarDouble { value:  double; }

table NTScalarArrayByte   { value: [ byte];   }
table NTScalarArrayUByte  { value: [ubyte];   }
table NTScalarArrayShort  { value: [ short];  }
table NTScalarArrayUShort { value: [ushort];  }
```

```
table NTScalarArrayInt    { value: [ int];    }
table NTScalarArrayUInt   { value: [uint];    }
table NTScalarArrayLong   { value: [ long];   }
table NTScalarArrayULong  { value: [ulong];   }
table NTScalarArrayFloat  { value: [ float];  }
table NTScalarArrayDouble { value: [ double]; }

union PV {
   NTScalarByte, NTScalarUByte, NTScalarShort, NTScalarUShort, NTScalarInt,
   NTScalarUInt, NTScalarLong, NTScalarULong, NTScalarFloat, NTScalarDouble,
   NTScalarArrayByte, NTScalarArrayUByte, NTScalarArrayShort,
   NTScalarArrayUShort, NTScalarArrayInt, NTScalarArrayUInt,
            NTScalarArrayLong, NTScalarArrayULong, NTScalarArrayFloat,
            NTScalarArrayDouble
}

struct fwdinfo_t {
   seq: ulong;
   ts_data: ulong;
   ts_fwd: ulong;
   fwdix: ubyte;
   teamid: ulong;
}

table fwdinfo_2_t {
   seq_data: ulong;
   seq_fwd: ulong;
   ts_data: ulong;
   ts_fwd: ulong;
   fwdix: uint;
   teamid: ulong;
}

union fwdinfo_u {
   fwdinfo_2_t,
}

table EpicsPV {
   name: string;
   pv: PV;
   timeStamp: timeStamp_t;
   fwdinfo: fwdinfo_t;
   fwdinfo2: fwdinfo_u;
}

root_type EpicsPV;
```

f142_logdata.fbs

```
include "fwdi_forwarder_internal.fbs";

file_identifier "f142";
```

```
table Byte   { value:  byte;   }
table UByte  { value: ubyte;   }
table Short  { value:  short;  }
table UShort { value: ushort;  }
table Int    { value:  int;    }
table UInt   { value: uint;    }
table Long   { value:  long;   }
table ULong  { value: ulong;   }
table Float  { value:  float;  }
table Double { value:  double; }

table ArrayByte   { value: [ byte];   }
table ArrayUByte  { value: [ubyte];   }
table ArrayShort  { value: [ short];  }
table ArrayUShort { value: [ushort];  }
table ArrayInt    { value: [ int];    }
table ArrayUInt   { value: [uint];    }
table ArrayLong   { value: [ long];   }
table ArrayULong  { value: [ulong];   }
table ArrayFloat  { value: [ float];  }
table ArrayDouble { value: [ double]; }

union Value {
  Byte, UByte, Short, UShort, Int, UInt, Long, ULong, Float, Double,
  ArrayByte, ArrayUByte, ArrayShort, ArrayUShort, ArrayInt, ArrayUInt,
  ArrayLong, ArrayULong, ArrayFloat, ArrayDouble
}

// Typical producers and consumers:
// Produced by EPICS forwarder from EPICS PV
// Consumed by NeXus file writer -> NXLog
// Consumed by Mantid -> Sample environment log
table LogData {
  source_name: string; // identify source on multiplexed topics, e.g. PV name
                       // if from EPICS
  value: Value;        // may be scalar or array
  timestamp: ulong;    // nanoseconds past epoch (1 Jan 1970), zero reserved
                       // for invalid timestamp
  fwdinfo: forwarder_internal;  // optional, for development and debug used
                                // by EPICS forwarder
}

root_type LogData;
```

f143_structure.fbs

```
// General schema which allows any EPICS structure to be forwarded as a
// flatbuffer. Generality comes at a price:  More overhead during construction
// in terms of space and cpu, more work for the receiver of the flatbuffer to
// access.
```

```
include "fwdi_forwarder_internal.fbs";

file_identifier "f143";

namespace f143_structure;

table Byte   { value:  byte;   }
table UByte  { value: ubyte;   }
table Short  { value:  short;  }
table UShort { value: ushort;  }
table Int    { value:  int;    }
table UInt   { value: uint;    }
table Long   { value:  long;   }
table ULong  { value: ulong;   }
table Float  { value:  float;  }
table Double { value:  double; }
table String { value:  string; }

table ArrayByte   { value: [ byte];   }
table ArrayUByte  { value: [ubyte];   }
table ArrayShort  { value: [ short];  }
table ArrayUShort { value: [ushort];  }
table ArrayInt    { value: [ int];    }
table ArrayUInt   { value: [uint];    }
table ArrayLong   { value: [ long];   }
table ArrayULong  { value: [ulong];   }
table ArrayFloat  { value: [ float];  }
table ArrayDouble { value: [ double]; }
table ArrayString { value: [ string]; }

union Value {
   Byte, Short, Int, Long, UByte, UShort, UInt, ULong, Float, Double, String,
   Obj, ArrayByte, ArrayShort, ArrayInt, ArrayLong, ArrayUByte, ArrayUShort,
   ArrayUInt, ArrayULong, ArrayFloat, ArrayDouble, ArrayString, ArrayObj,
}

table ObjM {
   k: string;
   v: Value;
}

table Obj { value: [ObjM]; }
table ArrayObj { value: [Obj]; }

table Structure {
   name: string;
   value: Value;
   timestamp: ulong;
   fwdinfo: forwarder_internal;
}

root_type Structure;
```

7.2.3 Event Formation Unit

The Event Formation Unit (EFU) receives raw data from the detector readout system and forms events as timestamp and pixel identifier pairs. It is available from the ess-dmsc event-formation-unit repository [9]. Its development is covered by BrightnESS task 5.1: *Creating a standard neutron event data stream for different detector types*. In this document, we are mainly interested in its role as a producer of data to the Kafka cluster. More detailed information about the EFU and the event formation task in general can be found in the BrightnESS deliverable 5.2: *Report processing choices for detector types*.

Raw data from the detector readout system is sent to the EFU over a dedicated fibre network. Each detector panel can be connected to one separate EFU, allowing the system to scale as needed. When an instrument adds detectors and panels, each new panel can be connected to a new EFU, which will process data in parallel with the existing units and forward the resulting event data to Kafka. This is shown in Figure 6.

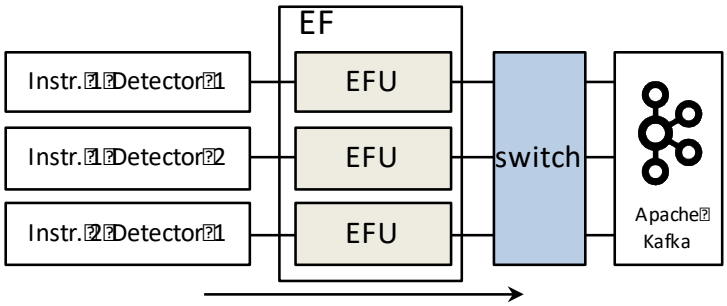

Figure 6. Event Formation Units producing data to Apache Kafka.

Tasks in the EFU are distributed among an input, a processing and an output thread. The first receives data from the detector readout system, while the second performs the event formation. The last thread then serialises the events obtained from the raw data using the FlatBuffers *ev42_events.fbs* schema and sends them to a Kafka cluster.

As an estimation for steady ESS operation with 16 instruments one should consider two instruments operating at 2×10^8 and 14 instruments at 5×10^6 events per second. With 64 bits per event as in the *ev42* schema, the expected average rate is approximately 2.2 GB/s of event data for the 16 instruments combined. From measurements of producer data rates around 200 MB/s on Data Management group servers, a number between 10 and 15 Kafka brokers would suffice to handle such a data rate. The machines used for the estimates are standard computers with relatively slow storage media, so higher data rates can be expected from more powerful servers, thus resulting in a smaller number of nodes required for the cluster. As mentioned above, due to the parallel nature of the event formation task, more nodes can be added to the Kafka cluster, should the instrument data rates increase. More detailed tests and analyses will be performed to specify the final Kafka cluster requirements, including optimisation of the cluster configuration on dedicated high-performance servers.

7.2.4 Fast Sample Environment

Besides detectors, instruments may include other high-rate data sources in the form of sensors measuring e.g. electric and magnetic field strengths. The acquisition of this type of data is covered by the Fast Sample Environment task under BrightnESS Work Package 5 (Task 5.2). This system will also produce data to the Kafka cluster, either using forwarded EPICS process variables or through an internal Kafka client. Expected data update frequencies are of the order of 1 kHz to 4 MHz, as mentioned in the EPICS section below,

resulting in data rates of up to about 61 MB/s per sensor. More information on this topic will be available in the upcoming deliverable 5.4.

7.2.5 EPICS Forwarder

The EPICS Forwarder, developed as part of task 5.3, is responsible for monitoring EPICS data sources (process variables) and for publishing the updated values of these data sources to the Kafka cluster. The source code can be found at the ess-dmsc forward-epics-to-kafka repository [10]. The data acquisition system has to be able to read updates from EPICS data sources in order to provide their values to the analysis software and the data persistence components, for example the NeXus File Writer.

EPICS

EPICS, the Experimental Physics and Industrial Control System [11], is a set of tools for building distributed control systems used by many large-scale scientific facilities around the world, such as particle accelerators and telescopes. It will be used by the Integrated Control System (ICS) division at ESS to provide access to *process variables* (PVs) that can be used to monitor and control accelerator and instrument devices. In the instruments, examples of devices that will have EPICS interfaces are motors, the sample environment and detector slow controls.

The ECP will interface to these EPICS devices to perform experiment control tasks. From the data acquisition point of view, EPICS PVs are sources of timestamped data to be aggregated and later written to file. The data aggregation and streaming applications could also expose their control interface using EPICS, although this is not currently planned.

Protocols to be used are determined by the EPICS version; EPICS 3 uses Channel Access, while EPICS V4 uses a new protocol named pvAccess. C/C++ libraries to use these protocols are available as part of the EPICS software distributions. Supporting Channel Access in addition to pvAccess allows the Forwarder to be tested at the existing facilities at the Paul Scherrer Institut (PSI) and ISIS. The EPICS Forwarder serialises the data updates from EPICS to FlatBuffers and delivers them into the messaging system based on Apache Kafka. Figure 7 shows the role of the Forwarder in the system architecture, obtaining EPICS data from an Input/Output Controller (IOC) and an EPICS V4 PVAServer.

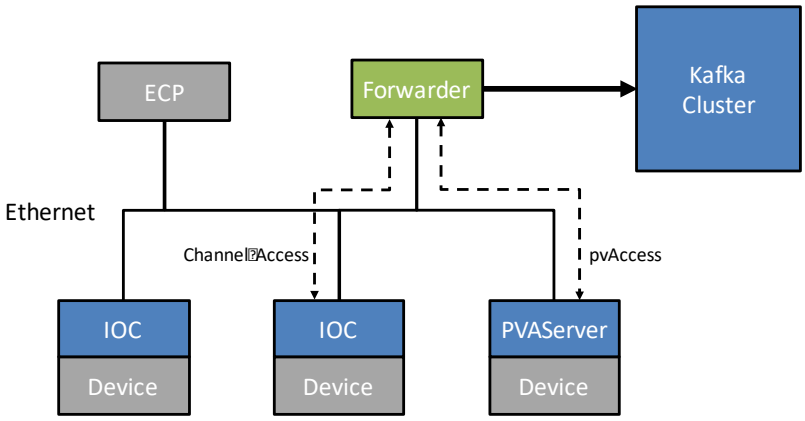

Figure 7. EPICS Forwarder and Apache Kafka.

Evaluation of the EPICS update frequency

Performance tests were done at PSI on a dedicated network using 10 Gigabit Ethernet, where two machines are available for testing purposes. With the official Apache Kafka performance

test tool and a custom test tool based on librdkafka, it was determined that there is a negligible difference between running the Kafka producer and the Kafka broker on the same versus on two separate machines in terms of total throughput. The following tests therefore show a configuration with the EPICS IOC running on one machine, and the EPICS Forwarder and the Kafka broker running on the second machine.

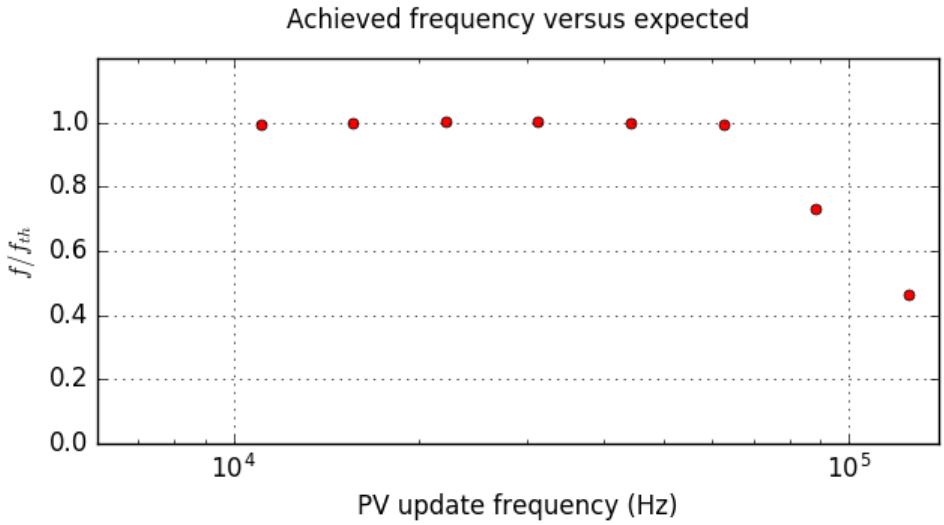

Figure 8. Fraction of the nominal EPICS update frequency received by the Forwarder and sent to Kafka.

We see in Figure 8 the achieved fraction of the nominal EPICS update frequency. The PV update frequency is set to the rate given on the x-axis, whereas the y-axis shows the fraction of that frequency at which the updates are received by the Forwarder and forwarded to Kafka. We see that the maximum is at about 65 kHz; the limiting factor here is the EPICS sub-system, where the limitation is not simply because of CPU load but rooted in the EPICS networking stack.

In Figure 9, we see the ratio of the actual update frequency and the set update frequency plotted against the total number of PVs for a different setup: in red circles, each PV is fixed at 1 kHz; in blue squares, each PV is fixed at 100 Hz. The PV payload size is chosen for a total nominal transfer rate of 150 MB/s in both of the data series. The graph shows how the performance of the EPICS Forwarder scales with the number of monitored PVs. As expected, more slower PVs can be forwarded (blue squares) before the limit is reached.

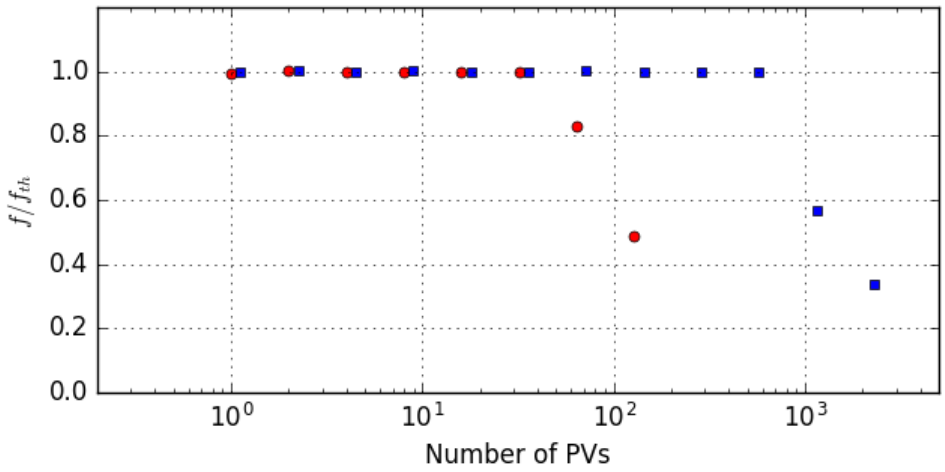

Figure 9. Forwarded PV update frequency against number of PVs for update rates of 1 kHz (red circles) and 100 Hz (blue squares), for a total nominal transfer rate of 150 MB/s.

Figure 10 shows the same data series, but plotted against the total update frequency. We see that also for this setup, the maximum forwarding frequency is compatible with the frequency that was found using only one PV (Figure 8).

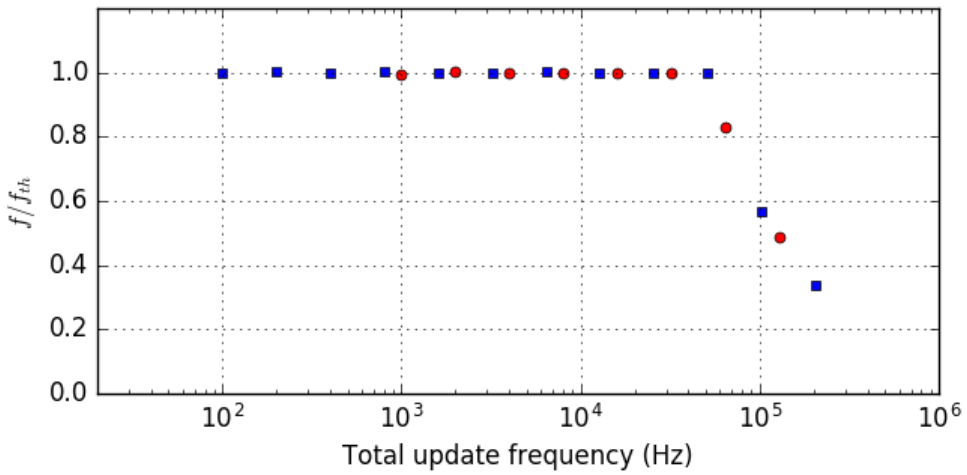

Figure 10. Forwarded PV update frequency against total update frequency for update rates of 1 kHz (red circles) and 100 Hz (blue squares), for a total nominal transfer rate of 150 MB/s.

The conclusion is that the EPICS Forwarder exceeds the performance of the EPICS source system both in terms of update frequency as well as transfer rate such that not only slow sample data can be transferred as originally planned, but also faster data like imaging data. Many sensors, like temperature and position sensors, produce data well below 100 Hz. The limit in raw update speed of EPICS can be avoided by batching multiple samples in one EPICS update. On the Kafka side, the Kafka protocol takes care of adaptive batching.

Functionality

The following subsections describe the main functionality of the EPICS Forwarder.

Monitor EPICS process variables

Using the EPICS V4 C++ library, the EPICS Forwarder monitors a given list of EPICS PVs for updates. The list of monitored PVs can be modified at runtime by sending command messages. By using the EPICS V4 library, the more recent EPICS pvAccess protocol as well as the older Channel Access protocol are supported.

Serialisation of EPICS data into FlatBuffers

When the EPICS Forwarder receives an update for a monitored PV, it will serialise the value into a FlatBuffers buffer. The schema to be used for serialisation is given by the configuration; the schemas are taken from the common *streaming-data-types* repository. Currently, the user can choose out of the box between the schemas *f142* for PVs which represent scalar or array numerical data, and schema *f143* which can represent an arbitrary hierarchical EPICS PV. Other more specialised schemas can be added on demand. Each update of an EPICS PV results in a FlatBuffers message. The Kafka protocol is designed to optimise the transfer of small messages at high frequency by e.g. automatic and adaptive batching.

Multiple serialisations per channel

It is possible to configure multiple serialisers with different FlatBuffers schemas per EPICS PV. This allows for sending of the same update in different formats to potentially different topics. The difference is not only in the FlatBuffers schema though, but different serialisers can also choose to convert only a certain subset of the data. It is also possible to send the same update to multiple Kafka clusters.

Parallelisation

On reception of an EPICS update event, the updated value is enqueued in the EPICS Forwarder and then distributed for serialisation over a configurable pool of worker threads. This architecture decouples the event processing layer of the EPICS V4 library from the remaining EPICS Forwarder and allows using many-core CPUs efficiently.

Extensibility

Support for new or specialised FlatBuffers schemas can be added in the plugin-like architecture without modifying the core code. The flexible configuration system also allows the FlatBuffers serialisers to be adapted to new use cases in the future.

Interfaces

The EPICS Forwarder can be run simply from a terminal or using common daemon frameworks like systemd [12]. On start-up, the software can be controlled using command-line arguments and a configuration file. Once started, further commands can be sent to the EPICS Forwarder in form of JSON messages via the Kafka topic which has been configured at start-up of the EPICS Forwarder.

Status and error conditions are published to a Kafka topic in JSON [13] format. More classic log messages are printed to *stdout* and optionally to a Graylog [14] server in the form of *Graylog Extended Log Format* (GELF) messages via a Kafka broker, or via the graylog-logger library developed at DMSC [15].

Software Architecture of the EPICS Forwarder

The EPICS Forwarder consists of a set of components, e.g. classes, functions. The main areas of responsibility are:

- Monitoring of EPICS PVs and enqueuing of EPICS updates in the work queue
- Serialisation worker thread pool and scheduling
- Connection to the Kafka cluster and publication of serialised EPICS updates
- Serialisation of EPICS updates to FlatBuffers. This is handled by serialiser plugins
- Monitoring and processing of JSON command message from a Kafka topic
- Publication of status information to the Kafka cluster
- Logging

Unit and Integration Tests

The open source googletest [16] library is used to facilitate unit testing. Unit tests are provided to test and validate the functionality of the software. The tests cover functionality which is internal to the program, like routines for accessing and manipulating data structures, but also certain cases which involve network I/O. Those tests which depend on external network services are opt-in. The interoperability with the full software stack is tested and verified in the integration tests. The software is profiled and tested for memory safety using Valgrind [17].

Documentation

The documentation is formatted using the commonly used open source tool Doxygen [18]. An overview, installation and usage examples are given in the project's README file, available at the code repository.

Installation and Usage

Dependencies

The EPICS Forwarder depends on the EPICS V4 (C++) and EPICS 3 libraries. It uses librdkafka for the communication with the Apache Kafka broker and data messages are serialised as FlatBuffers.

Command messages, status messages and configuration data are encoded as JSON which is a commonly used human-readable data format. A large number of available libraries make it easy to use JSON from almost any programming language. RapidJSON [19] is used in the EPICS Forwarder to work with JSON data in an efficient way. RapidJSON is a header-only C++ library and open source. A small number of additional open source supporting libraries can be viewed at the project's official README. The EPICS Forwarder is written in C++11 and built on GCC [20] and clang [21], using CMake [22].

Conclusion

The EPICS Forwarder has been developed as part of the project. It is a tested and working component of the data acquisition infrastructure and well exceeds the performance requirements evaluated so far. Development within WP5 continues to improve the usability and to add new features, like improved reporting of the current forwarding performance, more dynamic scheduling of the serialisation work queue and addition of support for more specialised FlatBuffers schemas.

7.2.6 NeXus File Writer

The NeXus File Writer is another new software development started by WP5. It is responsible for receiving the streamed data from the Kafka cluster and writing the data to NeXus compliant HDF5 files. First we briefly describe the underlying technologies before discussing the NeXus File Writer software in detail.

HDF5 and NeXus

The hierarchical data format HDF5 [23] is a data format developed originally for supercomputer applications at the National Center for Supercomputing Applications in the United States, and is today actively developed and used in a broad range of data-intensive industries and sciences. Even though HDF5 is designed as a format for single files, it is possible to link multiple HDF5 files together to form a larger logical HDF5 file. Version 1.10 of the HDF5 library is currently being used. The NeXus file format is a data schema built on top of HDF5 to unify the data storage and exchange of data from neutron, X-ray, and muon scattering experiments.

Functionality

The following subsections describe the main functionality of the NeXus File Writer.

Initiate File Writing on JSON Command

The File Writer will monitor the Kafka topic which is given at start-up for incoming command messages. File writing is initiated by a JSON command message on this Kafka topic. Each NeXus file is written by a separate set of threads. The NeXus File Writer process can therefore write multiple NeXus files in parallel.

The command message must specify the basic HDF5 group structure to be written to the NeXus file. It must also contain the list of data streams, identified by the topic and the source-name, which are to be written to the NeXus file. The NeXus File Writer then generates the basic HDF5 file hierarchy and connects to the required Kafka topics.

Write Data Messages to File

During file writing, the software is connected to the Kafka cluster and monitors the Kafka topics for messages which are supposed to be written to the NeXus file. Upon reception of the first data messages from a specified data stream, the FlatBuffers buffer contained in the message is inspected and an appropriate data stream writer for that FlatBuffers schema is created. Further data messages from the same data stream are passed to the same data stream writer.

The data stream writers know how a certain FlatBuffers schema is to be written to the HDF5 file and can be further customised via configuration options in the NeXus File Writer configuration file, as well as via options in the command message which initiates the file writing. The modular design of having separate data stream writers for each FlatBuffers schema decouples the system and allows for a plugin-like extensibility.

Interfaces

The NeXus File Writer can be run from a terminal or using common daemon frameworks like systemd. On start-up, the software can be controlled using command-line arguments and a configuration file. Once started, further commands can be send to the NeXus File Writer in form of JSON messages via the Kafka topic which has been configured at start-up of the NeXus File Writer.

Status and error conditions are published to a configurable Kafka topic in JSON format. More log messages are output to *stdout* and optionally to a Graylog server in the form of Kafka GELF messages via an intermediate Kafka broker, or via the graylog-logger library developed at DMSC. The usage of an intermediate Kafka broker decouples the system further and reduces the external dependencies.

Software Architecture of the NeXus File Writer

The NeXus File Writer consists of a set of components, e.g. classes, functions. The main areas of responsibility are:

- Connection to the Kafka cluster(s)
- Receiving messages from the Kafka cluster and distributing them among the Data Stream Writer Plugins
- Unpacking of the FlatBuffers messages and the writing of the data to HDF5 files by the Data Stream Writer Plugins
- Monitoring and processing of JSON command message from a Kafka topic
- Publication of status information to the Kafka cluster
- Logging

Unit and Integration Tests

The open source googletest library is used to facilitate unit testing. Unit tests are provided to test and validate the functionality of the software. The tests cover functionality which is internal to the program, like routines for accessing and manipulating data structures, but also certain cases which involve network I/O. Those tests which depend on external network services are opt-in. The interoperability with the full software stack is tested and verified in the integration tests.

The tests include a case where mock Kafka messages are used to initiate the writing of a file, upon which mock messages are being issued to serve example data to the File Writer. After the File Writer has written the data messages, the final HDF5 file is inspected and checked for correctness. The software is profiled and tested for memory safety using Valgrind.

Documentation

The main form of documentation is provided in form of comments in the source code as this keeps documentation and code most closely together. The documentation is formatted using the commonly used open source tool Doxygen. An overview, installation and usage examples are given in the project's README file, available at the code repository.

Installation and Usage

Dependencies

The NeXus File Writer uses librdkafka for the communication with the Apache Kafka broker. The data messages are serialised as FlatBuffers and for the writing of the data files, the official implementation of the HDF5 standard is used.

Command messages, status messages and configuration data are encoded as JSON which is a human-readable and human-writable data format and very commonly used in all kinds of software. A vast number of available libraries make it easy to use JSON from almost any

programming language. RapidJSON is used in the NeXus File Writer to work with JSON data in an efficient way. RapidJSON is a header-only C++ library and open source.

A small number of additional open source supporting libraries can be viewed at the project's README file. The NeXus File Writer is written in C++11 and compiled with GCC and clang using the CMake build system.

Performance of the NeXus File Writer

The whole data acquisition system has been designed with the performance requirements in mind. The transfer of the data as FlatBuffers is an important ingredient as it enables the NeXus File Writer to process the incoming data and to issue the disk I/O with very low computational overhead.

Performance measurements on the test system show a write speed which is compatible with the maximum write speed of the storage devices. The performance was measured with the incoming data messages being already in RAM so that the measurement is not impacted by the speed of the Kafka cluster but instead shows the speed of the file writing itself, meaning the performance of the HDF5 library and the physical storage device. The measurements are presented in Figure 11.

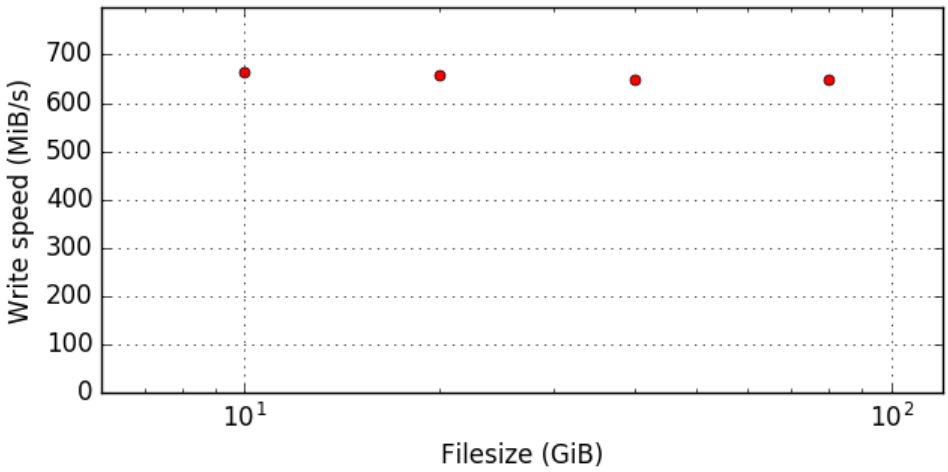

Figure 11. NeXus File Writer write speed against file size.

Conclusion

The task has developed a basic working NeXus File Writer. The code is tested and the write performance meets or exceeds expectations and requirements. Further development is concerned with support for the HDF5 Single Writer Multiple Reader feature, and the improvement of the integration with the Experiment Control Program.

7.2.7 Other Components

In principle, other data sources can be integrated into the architecture. It is expected however to see more subscribers to the data stream in addition to the file writer. The one subscriber that is already being integrated, is Mantid. Mantid is an open source software project for neutron instrument data reduction, developed and used by several facilities, including ESS [24]. It can be used offline, but is undergoing development for implementing live data reduction

functionality, allowing it to provide feedback to the instrument user in form of live visualisation. In this role, Mantid is a consumer of aggregated data, i.e. it subscribes to Kafka topics and deserialise FlatBuffers messages for consumption. Prototypes of this are already working; developed at STFC as in-kind partner, outside of the BrightnESS project, but in very close collaboration. This close integration of raw data taking and visualisation of reduced data results in tighter feedback cycles for experiments for the scientific users and is one of the main attractions of the architecture.

In addition to that visualisation utilities that consume event messages from the Kafka cluster have been developed to assist in understanding detector (prototype) data, especially in the context of the Event Formation Unit development and instrument commissioning. These utilities can generate a three-dimensional visualisation of a multigrid detector, with colour map information representing events.

7.3 Integration with External Components

The data aggregation software will interact with external systems; some of them are being developed by the Data Management Group and partners, such as the Data Curation System, while others, like EPICS and the Experiment Control Program, are under the responsibility of other ESS organisational units.

Each external system with which the aggregator must interoperate establishes an interface, where a protocol or workflow needs to be agreed upon. Some of the interfaces, such as with the EPICS systems, constrain communication to already defined protocols like Channel Access and pvAccess. Others do not have these limitations regarding technologies and protocols, as is the case of the experiment control system.

In this section, we look at the different external components with which interaction is expected, constraints and alternatives being considered for the integration with each of them.

7.3.1 Experiment Control

The Experiment Control Program (ECP) is the instrument user interface to the experiment. It allows the user to control devices in the beamline at a suitable level of abstraction, also permitting measurements to be scripted and orchestrated with sample environment condition changes. NICOS [25], a client/server experiment and instrument control system developed at the Heinz Maier-Leibnitz Zentrum (MLZ), has been selected as the platform for experiment control at ESS.

Possible interaction between the ECP/NICOS and data aggregation and streaming applications involve starting and stopping data acquisition, adding or removing sources of data, and controlling file writing. Alternatives for the interaction include communication through Kafka topics and EPICS. While Kafka allows for asynchronous publish-subscribe messaging, EPICS provides a request-response pattern and will already be a dependency of the ECP. Prototype work has been done, successfully sending commands from NICOS to start and stop the NeXus File Writer through Kafka.

7.3.2 Configuration

The data aggregation and streaming applications consist of generic software that can be deployed to any of the different ESS instruments (or partner facilities). Configuration that is specific to each of the instruments is kept separately from the applications' core code and must be obtained during initialisation and upon configuration changes. Some of the applications read data from files, while others accept commands with parameters through Kafka topics and TCP connections. A client library has been developed to provide a common

API for configuration through Redis [26]. Examples of application configuration include the addresses of Kafka brokers to use, sources of data to aggregate, and NeXus file structure for the File Writer.

ChannelFinder is a directory server with a REST API, developed for EPICS [27]. It allows channel names (e.g. PV names) to be stored in a database with associated user-defined tags and properties. Using the RecSync EPICS record synchroniser [28], the database can be automatically populated with PV names and status on IOC start-up. ChannelFinder can be used as a source of configuration to programs that work with EPICS PVs.

7.3.3 Data Curation System

The Data Curation System comprises a set of tools for cataloguing and providing access to experiment data and metadata, in accordance with the data policies in place. Interaction between data aggregation and streaming applications and the Data Curation System include obtaining metadata from experiment data files, and making these files available to users. Activities on the data curation task have officially begun recently and therefore there is no agreed standard for communications yet. One possibility is that interaction could occur through Kafka topics.

Non-experimental metadata user information and proposal to identifiers, which will reside in the User Office software, will need to be retrieved. The User Office software development has not started yet, hence the Data Curation System could form a proxy for that information for example with its REST API.

7.4 Current Tests

All the software components that are part of the data aggregation and streaming pipeline are being developed collaboratively by the partners as open source projects. These projects are continually built on a build server, where tests are run and binary artefacts are generated for deployment. Configuration of the machines in the build and test environment is done using Ansible [29], with the scripts kept under version control.

The applications being developed are tested both individually (as described in their respective sections) and in an integrated setup, to ensure they fulfil their requirements and work together in a production-like environment. In this section, we look at the different tests being run, focussing on the integration test. Build and test results are displayed on a large monitor at the DMSC office, close to the Data Management Group area, shown in Figure 12. This allows anyone interested to have an overview of the status of the different projects.

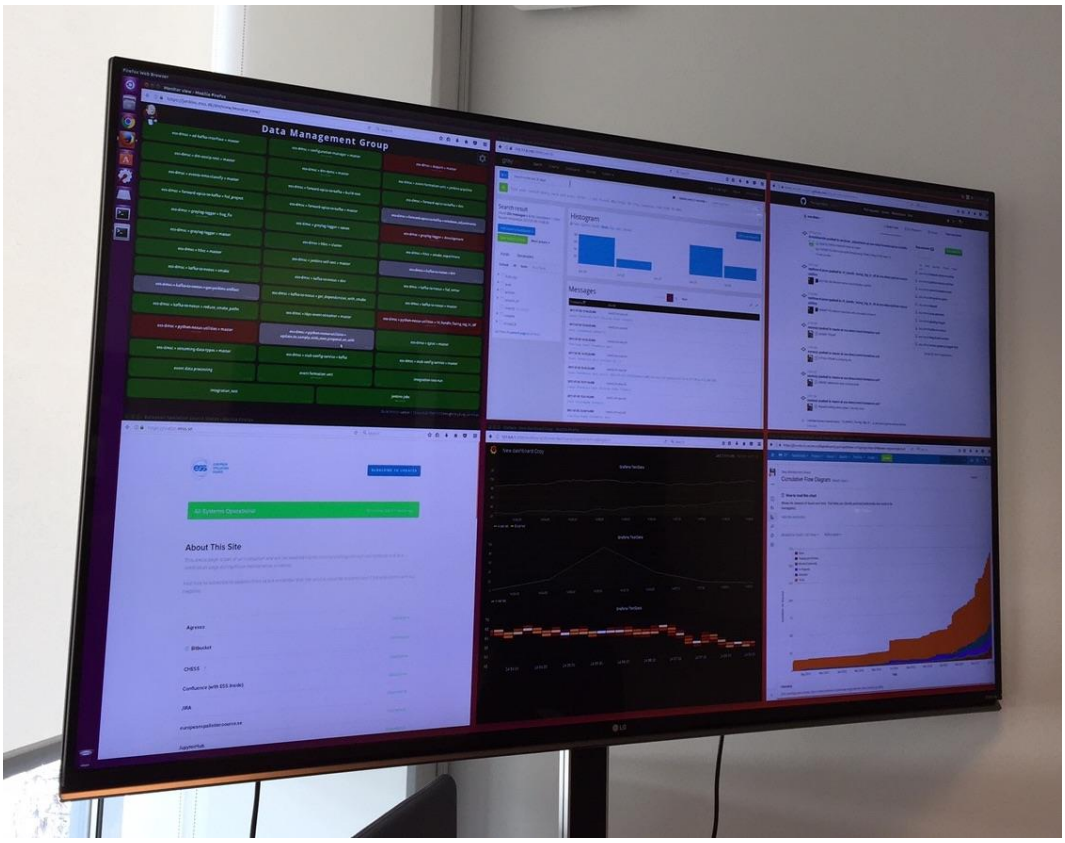

Figure 12. Project status monitor at the DMSC office.

7.4.1 Commit stage tests

A Jenkins build server automatically triggers builds and runs commit stage tests each time new code is pushed to a repository. The tests that are run at this stage vary according to the application but for C++ code in general include unit tests with googletest, static analysis with Cppcheck [30], test coverage reports with gcovr [31], code format compliance checking with clang-format [32] and detection of memory management problems with Valgrind.

These tests provide quick feedback to the developer, allowing problems to be discovered and fixed shortly after their introduction. With frequent commits, this results in a short feedback cycle, making it easier to locate the source of problems, as each commit causes a limited amount of change to the codebase. The tests can also be run by developers on their local machines before commits.

7.4.2 Integration test

An integration test was established during the BrightnESS MS32: *1st Integrated design review* [33] milestone meeting and has been running in the DMSC computing infrastructure. In this end-to-end test, data is sent from simulated sources and pre-recorded experiment files to an Apache Kafka cluster. The EFU, the EPICS Forwarder and the AreaDetector Kafka Plugin [34] run as data producers, while the NeXus File Writer consumes data from the topics and writes them to file. Values recorded in the resulting file are compared with expected values. Ansible is used to deploy software to the integration test nodes and to orchestrate the test, starting and stopping the applications and sending commands to them.

Three virtual machines have been configured to run the integration test. One of them is set up as a Jenkins node and is used to orchestrate the test using Ansible. This node deploys new

[35]versions of the applications and then executes an Ansible playbook with the test commands. This setup will be expanded to use more test machines, providing more isolation between applications in a more production-like environment, where each system component will be deployed to a different machine.

The integration test Kafka cluster has been set up with two broker nodes. The test uses data obtained from the multigrid detector prototype operating at the Spallation Neutron Source's CNCS instrument [35] [36]. These data are available on one of the integration test nodes and are fed to the Event Formation Unit for the neutron events to be calculated and sent to Kafka. EPICS data is being generated by a SINQ AMOR instrument simulation. ChannelFinder has also been added to the integration test. It receives the PV lists from the IOCs and supply status information such as whether the IOC is active. This installation allows evaluating ChannelFinder for configuring the data aggregation and streaming applications in the future.

Some applications, such as the EPICS Forwarder and the File Writer currently use JSON for configuration and commands. Part of the configuration is kept in files and loaded at start time, while other configuration is sent through special Kafka configuration topics.

The tests help discovering bugs in the applications and issues with their integration. Jenkins runs the test automatically twice a day, and additional runs can be triggered manually. Metrics are sent to a Graphite server and visualised with Grafana [37], while logs are sent to a Graylog server in the test environment. Screenshots of the servers' web interfaces are shown in Figure 13 and Figure 14, respectively.

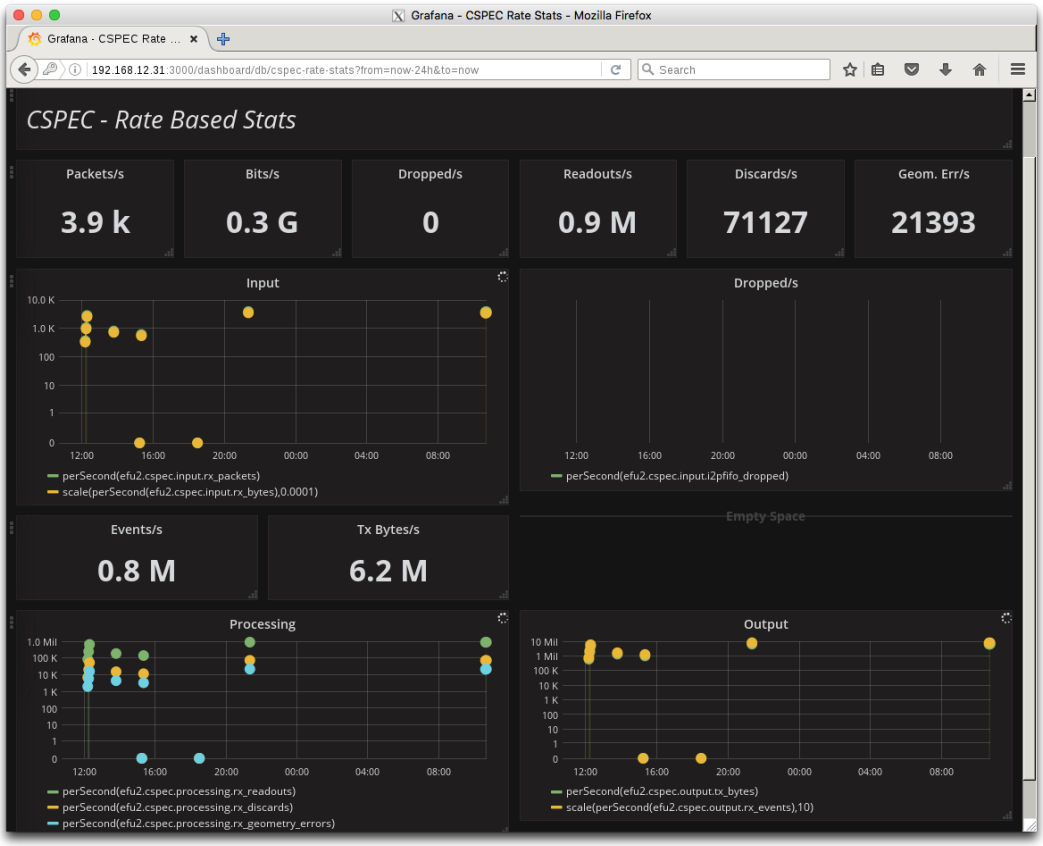

Figure 13. Grafana screenshot.

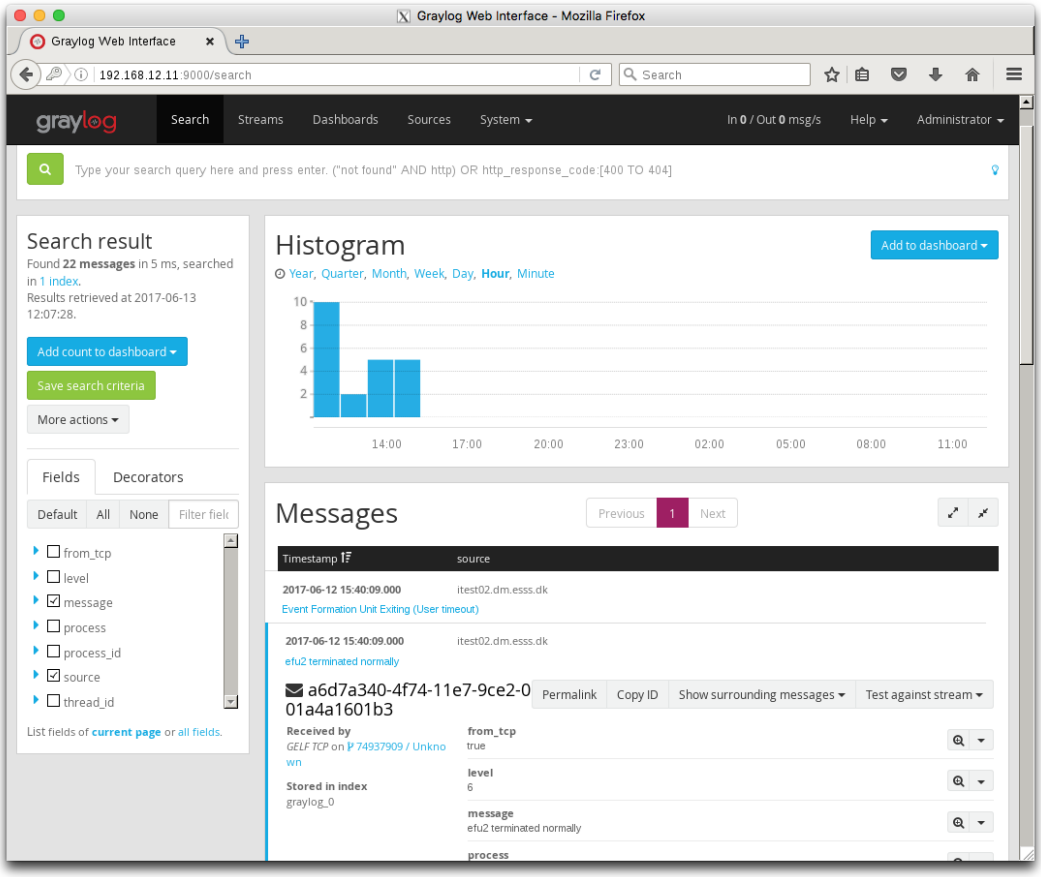

Figure 14. Graylog screenshot.

Latency and performance are not a concern in this virtualised environment. The main objective is to ensure that data correctly traverses the whole system.

Manual tests can be run on the ESS Instrument Integration Project (ESSIIP) laboratory. Three physical servers are installed there and connected through a 10 Gbps switch. The applications have also been deployed to that environment. These three servers are often used in different configurations to evaluate and optimise the performance of the software under development. Automation of these tests is planned when a target setup for the operations parameters and network configuration has been found. Among the activities performed was testing of the EPICS Forwarder and of the NeXus File Writer against PV data being served from multiple IOCs, including the mini-chopper IOC. These tests led to the identification and fixing of a number of problems in the software.

7.4.3 DonkiOrchestra

DonkiOrchestra is a workflow management framework for end-station software development created at Elettra [38]. It can provide logical organisation by sending a train of software triggers where each trigger activates some action according to different priority levels. This allows for concurrency and map-reduce strategies. Figure 15 shows the DonkiOrchestra architecture.

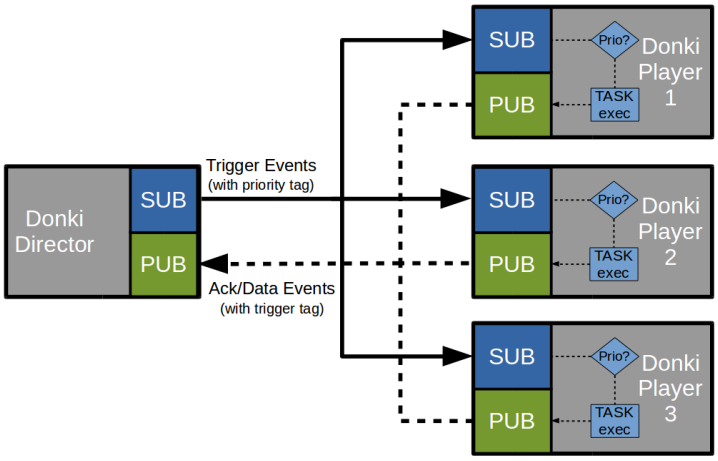

Figure 15. DonkiOrchestra architecture.

The system is composed of a Director and multiple independent Players. Each Player belongs to a Priority group and has a specific task to execute. The Director conducts the experimental sequence by sending a train of software triggers to the Players. For each step of the experimental sequence, a trigger signal is sent to the Players with the highest Priority (0), then to the group of Players with Priority 1 and so on. Each Player executes its task upon the arrival of the trigger and send back to the Director an acknowledge event. This is illustrated in Figure 16.

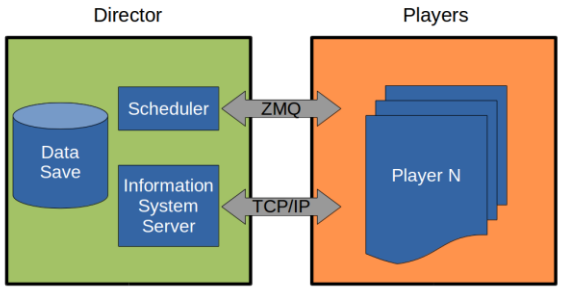

Figure 16. DonkiOrchestra Director and Players.

DonkiOrchestra has been repurposed to be used with Kafka and ESS technologies aiming at providing rapid scripting of complex scenarios of data stream generation and processing for simulation purposes. The framework became TANGO-independent [39] and a new information system based on TCP/IP protocols has been developed in order to establish communication between the Director and the Players. ZeroMQ [40] is the messaging system between the scheduler and the Players. This system should allow for partial testing and development of the data aggregation software.

7.5 Lessons Learned from Testing

The integration milestone tests identified a number of issues to be discussed. In this section, we look at these issues and the steps taken to address them.

Different approaches were being used to build and deploy the different applications under development. Some used Makefiles [41], other CMake; some had install targets, while others relied on manually copying files to the desired destination. Besides that, different levels of compiler warning tolerances raised issues when combining the applications and libraries. These differences are being settled by collaboration among the developers towards adopting CMake with standard target names and common standards for compiler settings.

Dependency management and deployment questions have also been identified. The lack of a standard for handling dependencies resulted in different approaches being used, such as cloning and building repositories at integration time, and deploying binary artefacts archived by Jenkins. Discussions have begun to standardise this, starting with documenting the existing approaches on Confluence [42], with their advantages and disadvantages. Package management solutions are also being investigated.

The multiple approaches to application configuration and control are being handled as part of a larger effort that involves deciding protocols and modes of interaction with the ECP, with the current approach being leaving application configuration in local files.

Server time synchronisation has also been identified as an issue; although neutron events and EPICS PVs are timestamped with information from the timing system, logs and metrics use the system time, this can cause problems when visualising and comparing them as they come from multiple nodes. This was addressed by enabling the Network Time Protocol (NTP) [43] on the nodes.

A number of application failure modes have been identified during the integration tests, and addressed by the developers of each of the applications. These lessons learned illustrate the success of the integration test at identifying problems that cannot be caught by the individual applications' commit stage tests. This includes catching problems arising from changes in the application interfaces.

7.5.1 Future Tests

The results of the integration milestone have been discussed in meetings at partner sites as well as between all WP participants and summarised in a Confluence page [44], raising a series of action items. These items establish the foundations for improvements to the integration test and future tests to be created.

One of the goals of the test improvements is to run the tests continuously, i.e. have the applications started and running for a long-time span; data can then be sent continuously, or alternatively be sent on a continual schedule triggered by Jenkins.

Currently the test uses a pre-recorded detector raw data file to feed data into the EFU. The test waits for all the file contents to be streamed, with some extra pause time to be sure that streaming can finish. The possibility of also limiting the test to a certain number of received events will be added, in order to better replicate the systems that will be used at ESS.

The existing checks on the streamed data are limited; only small subsets of the events from the recorded file and PV value changes are checked. As part of the test improvements, the generated NeXus file will be subjected to more extensive verification, both in terms of structure and data contents. Moreover, application status and response to valid and invalid commands and input can be tested; the current test only checks the status of the termination of the EFU.

In future tests, Kafka metrics will be automatically gathered and published in Grafana, allowing data rates per broker and per topic to be easily read and compared. This can also be used at the ESSIIP laboratory to measure performance.

Another issue identified was the desirability of being able to run integration tests in local virtual machines, to allow developers to run the tests on their own development machines before committing to the repositories.

The three physical servers in the ESSIIP laboratory will also be used for end-to-end automated performance tests, where the entire pipeline is deployed and run under realistic production-like conditions. These tests are not run on the testing environment on the DMSC computing infrastructure, as it is based on virtual machines that share the underlying computational resources, while the servers in the ESSIIP laboratory are dedicated physical machines. The shared nature of the computing resources, with the additional overhead of virtualisation, makes it unreliable to measure the absolute performance of applications in that environment.

Plans also exist to perform tests at the V20 test beamline at the Helmholtz-Zentrum Berlin (HZB), which is available to the ESS. NICOS has already been installed on it, operating with real instrument hardware. An initial test could be the addition of Kafka producer support to the detectors, as this would allow testing both the operation of a Kafka cluster and consumers such as the File Writer.

8 Outstanding Problems and Risks

The distributed nature of the system can cause problems, as it introduces modes of failure dependent on network connections and interactions among applications. It is especially important to be able to notice problems rapidly and provide some feedback to the user, if required. Debugging such a system can be more challenging than for a monolithic application running on one machine. Even deploying a patched version may require some orchestrated restart sequence for the entire pipeline. As the applications comprise a number of distinct programs, which will run on different instruments with differing configurations, a structured approach to deployment is needed. Part of this can be solved by using configuration management systems such as Ansible and Puppet [45]; the former is currently being used to deploy software to be tested to the integration environment. A discussion and evaluation of packaging and dependency resolution solutions is going on and will address this issue in more detail.

The tests are being performed and planned to introduce different expected failure modes and help identify them from their effects. Centralised logging with Graylog addresses part of the problem, as it allows obtaining information about the state of applications from a single log server. An online monitoring solution will also be set up for this purpose and for more general status information.

The complete pipeline needs to be tested under realistic production-like conditions. This will in the first instance be done in the ESSIIP laboratory. Any performance problems identified will result in a number of different measures to address them including tuning server parameters, changing the Kafka cluster topology and configuration, and adapting the data streaming and aggregation applications to distribute and parallelise processing.

Although tests in the integration and ESSIIP environment are of high importance for evaluating the software performance and identifying problems in general, they do not substitute for a full-scale test under the scrutiny of scientific users. Currently this is not possible to perform. At the V20 beamline at HZB tests are being planned to run under realistic, production-like conditions, against real instrument hardware on an operating facility. For this to be successful we need

to bring the integration with the experiment control in NICOS and data reduction within Mantid to maturity. For the construction phase the scope for these lies outside the DM group. However, in the current environment the interaction between the DMSC Instrument Data Group and their in-kind effort is very direct and fruitful. Face to face meetings are held at regular intervals and video conferences are held ad hoc quite frequently.

9 Conclusions and Roadmap

The entire data streaming system has been developed successfully. The whole pipeline is continually being tested in the automated integration setup, demonstrating that data can traverse the complete pipeline in a virtual environment. The schedule driven development that has been followed naturally brings up relevant topics in the regular integration meetings. While integration with external services (experiment control and Mantid for example) is not fully mature, good progress is being made in that area. This was not within the original scope of this deliverable.

Collaboration among developers at DMSC, BrightnESS and in-kind partners is facilitated by extensive use of the JIRA and Confluence web applications. Besides build and testing automation, Jenkins provides a central location for monitoring the status of the various projects.

One of the next steps in task 5.3 includes discussing and specifying the responsibilities and protocols that govern the interactions among data aggregation and streaming applications and external components, such as the ECP and the User Office, and also refining the interactions among the applications themselves.

Deploying the complete pipeline to the ESSIIP laboratory is another activity expected for the coming months. It will allow testing the performance of the system under realistic conditions and using a production-like network infrastructure. The laboratory also contains real instrument hardware, so activities will also benefit other ESS organisational units that interact with the aggregation software, allowing them to perform tests against it. Yet more integration will be demonstrated at the V20 beamline after successful completion of the ESSIIP exercise. This is planned before the end of the BrightnESS grant.

10 List of Publications

- Reis, C., Borghes, R., Kourousias, G., Pugliese, R. *A Simulation System for the European Spallation Source (ESS) Distributed Data Streaming*. Poster accepted for ICALEPCS 2017, Barcelona, Spain.
- Koennecke, M., Brambilla, M., Werder, D. *EPICS data streaming and HDF file writing for ESS benchmarked using the virtual AMOR instrument*. Poster accepted for ICALEPCS 2017, Barcelona, Spain.
- Mukai, A. H. C., Christensen, M. J., Nilsson, J., Richter, T. S., Shetty, M., Brambilla, M., Koennecke, M., Werder, D., Akeroyd, F. A., Clarke, M., Jones, M. *Status of the Development of the Experiment Data Acquisition Pipeline for the European Spallation Source*. Poster accepted for ICALEPCS 2017, Barcelona, Spain.

11 References

[1] Apache Kafka, [Online]. Available: http://kafka.apache.org. [Accessed 27 07 2017].

[2] ess-dmsc GitHub organisation, [Online]. Available: https://github.com/ess-dmsc. [Accessed 27 07 2017].

[3] “Jenkins,” 28 07 2017. [Online]. Available: https://jenkins.io.

[4] “5.1: Design report data aggregator software,” 08 2016. [Online]. Available: https://brightness.esss.se/about/deliverables/51-design-report-data-aggregator-software. [Accessed 27 07 2017].

[5] “librdkafka,” [Online]. Available: https://github.com/edenhill/librdkafka. [Accessed 27 07 2017].

[6] “NeXus,” [Online]. Available: http://www.nexusformat.org. [Accessed 27 07 2017].

[7] “FlatBuffers,” [Online]. Available: http://google.github.io/flatbuffers/. [Accessed 27 07 2017].

[8] “streaming-data-types,” [Online]. Available: https://github.com/ess-dmsc/streaming-data-types. [Accessed 27 07 2017].

[9] “event-formation-unit,” [Online]. Available: https://github.com/ess-dmsc/event-formation-unit. [Accessed 27 07 2017].

[10] “forward-epics-to-kafka,” [Online]. Available: https://github.com/ess-dmsc/forward-epics-to-kafka. [Accessed 27 07 2017].

[11] “Experimental Physics and Industrial Control System,” [Online]. Available: http://www.aps.anl.gov/epics/. [Accessed 27 07 2017].

[12] “systemd,” [Online]. Available: https://www.freedesktop.org/wiki/Software/systemd/. [Accessed 28 07 2017].

[13] “JSON,” [Online]. Available: http://json.org. [Accessed 27 07 2017].

[14] “Graylog,” 27 07 2017. [Online]. Available: https://www.graylog.org.

[15] “graylog-logger,” [Online]. Available: https://github.com/ess-dmsc/graylog-logger. [Accessed 27 07 2017].

[16] “googletest,” 27 07 2017. [Online]. Available: https://github.com/google/googletest.

[17] “Valgrind,” 27 07 2017. [Online]. Available: http://valgrind.org.

[18] “Doxygen,” [Online]. Available: http://www.stack.nl/~dimitri/doxygen/. [Accessed 27 07 2017].

[19] “RapidJSON,” [Online]. Available: http://rapidjson.org. [Accessed 27 07 2017].

[20] “GCC,” [Online]. Available: https://www.gnu.org/software/gcc/. [Accessed 27 07 2017].

[21] “clang,” [Online]. Available: http://clang.llvm.org. [Accessed 27 07 2017].

[22] “CMake,” [Online]. Available: https://cmake.org. [Accessed 27 07 2017].

[23] “HDF5,” [Online]. Available: https://www.hdfgroup.org/downloads/hdf5/. [Accessed 27 07 2017].

[24] “Mantid,” [Online]. Available: http://www.mantidproject.org/Main_Page. [Accessed 27 07 2017].

[25] “NICOS,” [Online]. Available: http://www.nicos-controls.org/. [Accessed 27 07 2017].

[26] “Redis,” [Online]. Available: https://redis.io. [Accessed 27 07 2017].

[27] “ChannelFinder,” [Online]. Available: http://channelfinder.github.io. [Accessed 28 07 2017].

[28] “RecSync,” [Online]. Available: https://github.com/ChannelFinder/recsync. [Accessed 27 07 2017].

[29] “Ansible,” [Online]. Available: https://www.ansible.com. [Accessed 27 07 2017].

[30] “Cppcheck,” [Online]. Available: http://cppcheck.sourceforge.net. [Accessed 28 07 2017].

[31] Gcovr. [Online]. Available: http://gcovr.com. [Accessed 28 07 2017].

[32] “clang-format,” [Online]. Available: http://clang.llvm.org/docs/ClangFormat.html. [Accessed 28 07 2017].

[33] “MS32: 1st Integrated design review,” [Online]. Available: https://brightness.esss.se/about/milestones/ms32-1st-integrated-design-review. [Accessed 28 07 2017].

[34] “ad-kafka-interface,” [Online]. Available: https://github.com/ess-dmsc/ad-kafka-interface. [Accessed 28 07 2017].

[35] A. Khaplanov et al., “Multi-Grid detector for neutron spectroscopy: results obtained on time-of-flight spectrometer CNCS,” *Journal of Instrumentation,* vol. 12, p. P04030, April 2017. *doi:10.1088/1748-0221/12/04/P04030*

[36] “4.5: Simulation and Generic multi-grid design,” [Online]. Available: https://brightness.esss.se/about/deliverables/45-simulation-and-generic-multi-grid-design. [Accessed 18 August 2017].

[37] “Grafana,” [Online]. Available: https://grafana.com. [Accessed 28 07 2017].

[38] R. Borghes and G. Kourousias, “DonkiOrchestra: a scalable system for data collection and experiment management based on ZeroMQ distributed messaging,” in *NOBUGS*, Copenhagen, 2016.

[39] “TANGO,” 28 07 2017. [Online]. Available: http://www.tango-controls.org.

[40] “ZeroMQ,” [Online]. Available: http://zeromq.org. [Accessed 28 07 2017].

[41] “GNU Make,” [Online]. Available: https://www.gnu.org/software/make/. [Accessed 28 07 2017].

[42] “ESS Confluence,” 28 07 2017. [Online]. Available: https://confluence.esss.lu.se.

[43] “NTP: The Network Time Protocol,” [Online]. Available: http://www.ntp.org. [Accessed 28 07 2017].

[44] “Integration Milestone 1: Problems, solutions and evaluation,” [Online]. Available: https://confluence.esss.lu.se/display/DMSC/Integration+Milestone+1%3A+Problems%2C+solutions+and+evaluation. [Accessed 28 07 2017].

[45] “Puppet,” [Online]. Available: https://puppet.com. [Accessed 28 07 2017].