brightness

BrightnESS

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

H2020-INFRADEV-1-2015-1

Grant Agreement Number: 676548

Deliverable Report: D5.1 Design report 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.: 5.1	
	Deliverable Type: Report	
	Dissemination Level: Public	Contractual Delivery Date: 31.08.2016
		Actual Delivery Date:
	EC Project Officer: Bernhard Fabianek	

2 Document Control Sheet

Document	Title: BrightnESS_Deliverable_5.1	
	Version: 0.1	
	Available at: https://brightness.esss.se	
	Files: 1	
Authorship	Written by	Afonso Mukai (WP5), Tobias Richter (WP5 leader)
	Contributors	Martin Shetty (WP5), Mark Könnecke (WP5), Dominik Werder (WP5), Michele Brambilla (WP5)
	Reviewed by	Roy Pennings (WP1), Raquel Costa (WP1)
	Approved by	Steering Board Members

3 List of Abbreviations

API	Application programming interface
BSD	Berkeley Software Distribution
DESY	Deutsches Elektronen-Synchrotron (Germany)
DLS	Diamond Light Source (UK)
DM	Data Management (Group in DMSC)
DMSC	Data Management and Software Centre
EPICS	Experimental Physics and Industrial Control System
ESS	European Spallation Source
ESSIIP	ESS Instrument Integration Project
HTML	HyperText Markup Language
ICS	Integrated Control System
IDL	Interface Description Language
IOC	Input/Output Controller
IPv6	Internet Protocol version 6
JVM	Java virtual machine
LGPL	GNU Lesser General Public License
NSLS	National Synchrotron Light Source (United States)
PSI	Paul Scherrer Institut (Switzerland)

PV	Process Variable
SASL	Simple Authentication and Security Layer
SLS	Swiss Light Source (Switzerland)
SSH	Secure Shell
SSL	Secure Sockets Layer
STFC	Science and Technology Facilities Council (UK)
TCP	Transmission Control Protocol
TDC	Top Dead Centre (chopper timing signal)
UDP	User Datagram Protocol

4 List of Figures

Figure 1. The data aggregator software in the ESS data acquisition architecture. 6
Figure 2. A typical EPICS system architecture. 11
Figure 3. ZeroMQ and EPICS v4 throughput for repeated event data. 11
Figure 4. ZeroMQ throughput for different PSI instrument data sets and message sizes. 13
Figure 5. ZeroMQ throughput for different numbers of clients. 13
Figure 6. A Kafka aggregation architecture in the ESS data acquisition. 15
Figure 7. Kafka topic and partitioned log. 15
Figure 8. Comparison of serialisation and compression tools. 18

Table of Contents

1 Project Deliverable Information Sheet 2
2 Document Control Sheet 2
3 List of Abbreviations 2
4 List of Figures 3
Table 3
5 Executive Summary 4
6 Report on Implementation Process and Status of Deliverable 4
7 Data Aggregation Technology Choices 5
7.1 Data Aggregation at ESS 5
7.1.1 The ESS Data Acquisition Architecture 5
7.1.2 Data Aggregation and Streaming 8
7.2 Data Streaming Technology Choice 9
7.2.1 Criteria 9
7.2.2 Technology Alternatives 10
7.2.3 Result and Conclusion 16
7.3 Technology Choice for Data Serialisation 16
8 Roadmap – Development, Testing, and Deployment 18
8.1 System Design and Architecture 18
8.2 The Apache Kafka Cluster 18
8.3 Data Producers and Consumers 19
8.4 Serialisation and Schema Management 20
8.5 Collaboration and Tools 20
8.6 Testing and Deployment 21
9 Risks and Mitigation Strategies 22
10 Conclusion 23
11 List of Publications 24

12 References 24

5 Executive Summary

The data aggregator software task in work package 5 is progressing as planned. We decided to use Apache Kafka as the underlying technology for aggregation and streaming, and Google FlatBuffers as the serialisation library. Development work is going on with an agile approach focussed on early software tests and delivery, addressing the requirements in iterations. Collaboration tools are being used to track the progress of the project, discuss issues and maintain documentation, with virtual meetings held every two weeks.

Software modules to generate simulated data streams for integration and performance evaluations are ready to be put into our testing and deployment infrastructure. This infrastructure consists of a build server and with virtual machines for deploying and running integration tests as well as a physical lab space with three servers to be installed at the ESSIIP laboratory in Lund, where software from the Data Management group can be tested and integrated with real hardware and software from other ESS groups.

In the coming development cycles we are going to validate the technology choices and system design and architecture in both settings, addressing problems that might be discovered in the tests. The use of Kafka in projects handling large volumes of data is an indication that it can satisfy our requirements. Results from the tests and future project steps will be documented in JIRA, Confluence and Bitbucket.

6 Report on Implementation Process and Status of Deliverable

This document is Deliverable 5.1 “Design Report for the Data Aggregator Software” for the H2020 BrightnESS project at the European Spallation Source (ESS).

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 and the recording in event mode, 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 processed for live reduction and visualisation, providing real-time feedback while the experiment is running.

The Data Management and Software Centre (DMSC) located in Copenhagen, Denmark, is responsible for the acquisition and analysis of the scientific data from the ESS neutron

instruments. The Data Aggregator 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 and to the user is the primary reason for coming to the facility.

The data aggregation task (5.3) 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 and Paul Scherrer Institut (PSI) in Switzerland. It deals especially with the integration of the high volume neutron data from the processing of events in task (5.1). The DM group also has an in-kind collaboration agreement with the Science and Technology Facilities Council (STFC) in the United Kingdom for any remaining data streaming tasks including file writing.

This report presents the current design for the data aggregator software system. The document discusses the data aggregation task, requirements for the system, the technology alternatives considered, and presents the design decisions made. The proposed design is not static and will evolve in an agile manner, as prototypes and early delivery and testing of versions of the software allow for a better understanding of requirements, interaction with the relevant stakeholders, and early discovery of problems in the development process. A roadmap with the approach to development, testing and deployment of the software system is also presented.

7 Data Aggregation Technology Choices

7.1 Data Aggregation at ESS

7.1.1 The ESS Data Acquisition Architecture

An overview of the system architecture for data acquisition at ESS is illustrated in Figure 1, where the items in different colour zones are under the responsibility of different organisational units. Items in the yellow box will be provided by the Neutron Instrument Technology Division, items in the blue box will be provided by the Integrated Control System (ICS) Division, and items in the green box plus the Event Formation, by the DMSC.

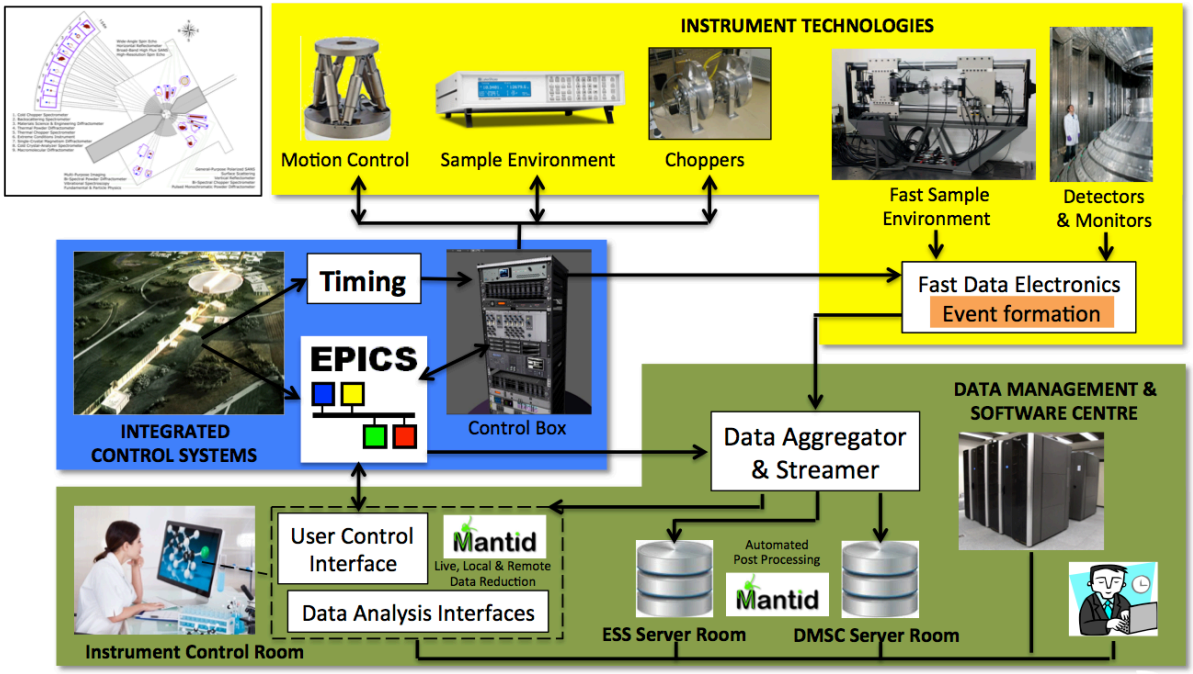

Figure 1. The data aggregator software in the ESS data acquisition architecture.

Some typical components of a neutron scattering instrument are shown in the yellow box: the choppers that select wavelength ranges along the neutron beam line, the sample orientation (motion control) devices that position the sample in the neutron beam, the temperature controller devices that control the temperature of the sample and the neutron detectors that are measuring the neutrons scattered from the sample. These instrument components are the responsibility of the Neutron Technologies Division in Science Directorate.

In terms of data input and output for these devices we can characterise fast data as data that is in synchronisation with the production of the pulses of neutrons by the accelerator, while slow data does not need to be. In general, but there are exceptions, the control data for the instrument components (i.e. the data values sent to the components asking them to do something) are slow data. An example of slow data would be a request to a temperature controller to change the sample temperature or to a chopper controller box to change the phase of the chopper. Examples of fast data would be the detector data and possibly fast sample environment data such as rapidly varying electromagnetic fields. In an intermediate category, an example of medium-fast data is the chopper top dead centre (TDC) signal, which indicates its position as it rotates. Table 1 shows indicative data rates associated with the different sources of data.

Table 1. Data source types.

Type	Examples	Rate	Characteristics
Slow	Motion control, sample environment	< 14 Hz average	Labelled monitor messages
Medium	Choppers, sample environment	< 20 kHz maximum	Buffered readout
Fast	Detectors, sample environment	> 20 kHz	Event-type message stream

The responsibility for providing the low level control to the instrument components rests with the Integrated Control System Division in the Machine Directorate. This will be done using the system known as EPICS (Experimental Physics and Industrial Control System) that is also being used for the controls of the ESS accelerator and target, as well as a large number of other large scale science facilities around the world. The Instrument Data group at DMSC is responsible for the high level experiment control, providing the user an appropriate scientific interface that abstracts away unnecessary technical details of the low level EPICS layer.

The timing signal from the accelerator will be passed to the instruments on an optical fibre network. It will be received on the instruments by the timing receiver card that can be programmed to send signals to the instrument components, shown as *Timing* in Figure 1. The communication from EPICS to the instrument components is via software known as IOCs (input/output controllers) that will run in a machine known as a control box, where the timing receiver card will also be. Each IOC provides a number of process variables (PV) with values and locally generated timestamps. Responsibility for the control box lies with the ICS Division.

The protocol used for control and feedback will be EPICS. pvAccess (EPICS v4) is the newer, less mature standard, the older EPICS 3 Channel Access is a reliable, established standard, but it does not work well for larger data sizes, like detector images, or for complex data structures or those that vary in size. For these types of data ICS are considering using EPICS pvAccess (v4) exclusively. In both protocols, EPICS PVs can be polled or monitored, so that they send an update when the value changes beyond a configurable threshold. We anticipate that monitoring will be the preferred way of operating.

As shown in Table 1, there are different rates for data sources. The slow data, e.g. the temperature of the sample, will be read via EPICS. The fast data from the neutron detectors will be read through an interface to the detector systems. In Figure 1 this is symbolically shown as the *Fast Data Electronics/Event* formation box. Medium-fast data, such as the TDC signals from the choppers, or fast sample environment data from pump-systems (e.g. electric fields, lasers, etc.) used in pump-probe experiments may be delivered using EPICS.

Except for the few image-based detectors, detector data at ESS will be acquired in *event mode*, i.e. each neutron counted in a detector will be recorded individually with both its position on the detector as a pixel identifier and the time at which it was recorded.

Table 2 shows the anticipated rates at detectors for some ESS instruments, according to analytical calculations, simulation and extrapolation from existing reference instruments.

Table 2. Anticipated global and local detector rates for some ESS instruments.

Instrument	Total rate on sample [n/s]	Global rate [Hz]	Average rate [Hz/cm^2]	Peak rate [Hz/cm^2]	Peak instantaneous rate [Hz/cm^2/ms]
BEER	10^9	2×10^5	70	700	17
C-SPEC	10^8	2×10^5	5	500	1000
DREAMS	3.4×10^8	10^7	110	10^3	24
ESTIA		100×10^6	2×10^6	10^7	
FREIA	5×10^8	12×10^6	2×10^6	10^7	
HEIMDAL	2×10^9	8×10^6	400	4×10^3	95
LOKI	$\leq 10^9$/cm^2	40×10^6		50×10^3	200
SKADI	$\leq 10^9$/cm^2	40×10^6		50×10^3	200

T_REX	10^8	2×10^5	5	500	1000

For the pixel identifier a 32-bit value may be sufficient, but some of the detectors (like on the NMX instrument) could have a spatial resolution and active area that is higher than what can be expressed in 32-bit; timestamps are likely recorded in 32 bits.

This data will be supplied to the *Data Aggregator* shown in Figure 1. The data aggregator represents both hardware and software running in real time on that hardware. The data aggregator will also receive any other fast data, such as chopper TDCs and fast sample environment data, and slow data (metadata) from the EPICS system. The slow data acquired from the instrument components will be time stamped in the IOC's with the absolute time at which it was measured. The data will then be aggregated together with the neutron data and other information relating to the pulse from the accelerator.

In Figure 1 the data aggregator is also labelled as the *Data Streamer*. The data that is aggregated in frames will be made available as a publish/subscribe network stream. A subscriber then connects over the network and receives the data as a stream from the data aggregator. An obvious subscriber for the data is the software that writes the data file; the data are then streamed in real time, essentially frame by frame, to the file writer software and the data files are written on the fly.

7.1.2 Data Aggregation and Streaming

At ESS, experimental data and metadata will be generated by multiple sources and transported using different serialisation formats and communication protocols. Neutron events from detectors, temperature and pressure readings from the sample environment, and chopper settings are examples of data that are of interest to a user performing an experiment. User and proposal information will be available from information systems and also need to be associated with experiment data.

On the client side, software systems performing tasks such as file writing and live data reduction and visualisation need to fetch data from the aforementioned sources. The data aggregator software abstracts the connection to them by providing a uniform interface to all different types of data. It offers the clients single channels through which data from multiple sources can be obtained.

Three subscribers to the stream from the data aggregator are shown in Figure 1. At least two of these will always be running, while the third will most probably be running all of the time. The two subscribers that would always be running are streaming file writers, which persist experiment data and metadata to files in the NeXus format; one writing to disk in the Lund server room and the other to disk in the DMSC server room in Copenhagen. The third subscriber to the data stream is the rack-mounted computer hosting the *User Control Interface* running the Mantid data reduction software; data reduction is the transformation of a dataset collected from an instrument into a dataset in physical units. Using the *Live Listener* part of the Mantid software it will receive the live data stream and process it on the fly (in real time).

The data aggregator software is a critical part of the ESS data acquisition architecture, as it will provide the channel through which clients obtain experiment data. It must be able to connect and retrieve data from the sources, which include EPICS servers (e.g. sample environment and motion control) and servers using other protocols over Ethernet; clients must be able to subscribe to the data they are interested in, which may be a subset of the

data being aggregated for a given instrument; the transport of data must be able to be sustained with appropriate performance, keeping up with the events from the neutron detectors, which are expected to be the sources with the highest data rates; and all of this is expected to be done using commercial, off the shelf hardware that can easily be deployed, maintained and replaced.

7.2 Data Streaming Technology Choice

7.2.1 Criteria

Section 7.1 discussed the role of the data aggregator software in the general ESS data acquisition architecture. That set of requirements might be satisfied by a range of alternative solutions, built with different technologies. In this section we present the criteria used to guide this decision, with examples of factors that can be used to evaluate the technologies against them.

7.2.1.1 Technical Suitability

The resources provided by the streaming technology determine whether it is technically adequate for satisfying the requirements of the aggregator. Among the technical factors considered are support for complex or custom data structures, availability of a suitable protocol, the possible buffering, caching and message delivery topologies, and configurable error handling (e. g. buffering, retrying, discarding messages). Availability on different platforms, support of IPv6 multicasts, security features such as authentication and vulnerabilities, reliability, concerning lost messages, handling of clients and sources connecting and disconnecting, and stability are also taken into account.

7.2.1.2 Performance

For handling the high rate of data expected from the high neutron flux at ESS, the streaming technology must allow for appropriate performance to be achieved. The maximum data rates allowed by it, as well as the response when under too high a data rate and the way errors are handled in this situation are relevant factors. The ability of the technology to scale affects the ability to handle increased data rates with upgrades and the addition of more data sources and clients.

7.2.1.3 Maintainability

ESS is expected to operate for a period comprising several decades. During development and afterwards, in commissioning and operations, the data aggregator software might need to be modified for bug fixes, addition of new functionality, or satisfying new performance requirements in face of facility upgrades. For this reason, maintainability is of paramount importance for the choice of streaming technology. Factors such as the level of ongoing development of the technology, indication of future direction, and expected longevity are taken into account in this evaluation. Other important factors are backward compatibility of version upgrades, software dependencies and the maintenance state of these packages, and also code readability and quality, as that might affect our ability to fix the code ourselves when necessary.

7.2.1.4 Ease of use

The ease of use affects how quickly developers can start to use and understand the technology. It is an important quality during development, as not all developers might be familiar with the technology, but also for the future, when new staff joins ESS and might

become responsible for maintaining and developing the data aggregator software. This quality can be evaluated by the complexity involved in creating simple and more advanced examples, the tools and knowledge required to build software and integrate it into the aggregator, the programming languages supported by it and the availability of documentation, including manuals and examples.

7.2.1.5 Popularity

Using a technology that is also employed by other facilities in the neutron and light source community allows us to exchange experiences and reach for collaboration and support, especially if it is being used in similar types of projects. Internally, the potential community in ESS can be evaluated, considering how the solution fits in or might be used by other ESS or DMSC projects and its suitability for scientific projects. The size of the general community outside large scientific facilities is also a good indicator of the amount of activity around the technology and mailing lists and forums provide a means to exchange support. Finally, the availability of commercial support might be helpful in case consulting or training be needed. The presence of commercial support is also a good indicator for the popularity of the code and gives a good impression of the overall health of the code and ecosystem around it.

7.2.2 Technology Alternatives

The alternative technologies considered for data streaming were EPICS v4, ZeroMQ, and Apache Kafka. In the following subsections, we describe and evaluate each of them against the criteria from section 7.2.

7.2.2.1 EPICS v4

EPICS is a set of software tools, libraries and applications for creating distributed soft real-time control systems, available under the EPICS Open license, and mainly used in large scientific facilities. The most widely used version is currently EPICS 3, which provides a protocol named Channel Access. With EPICS, a client can request the value of a PV through its name; UDP is used for resolving the channel name to the responsible server, allowing a TCP connection to be established, so that information can be exchanged using Channel Access. EPICS v4 builds upon EPICS 3 and adds support to structured data and a new protocol named pvAccess, among other functionality. At ESS, ICS is considering the use of EPICS v4 for slow control system software.

A typical EPICS system architecture is shown in Figure 2. Each client establishes a separate connection to each of the servers that is responsible for PVs it is interested in; this requires the clients to know the name of all the PVs they use. There are EPICS applications that can modify this architecture, such as the PV Gateway and ChannelFinder. The former allows many clients to access a PV making only one connection to a server, while the latter provides a directory server capable of associating properties and tags with the channel names that identify PVs, allowing queries to be made.

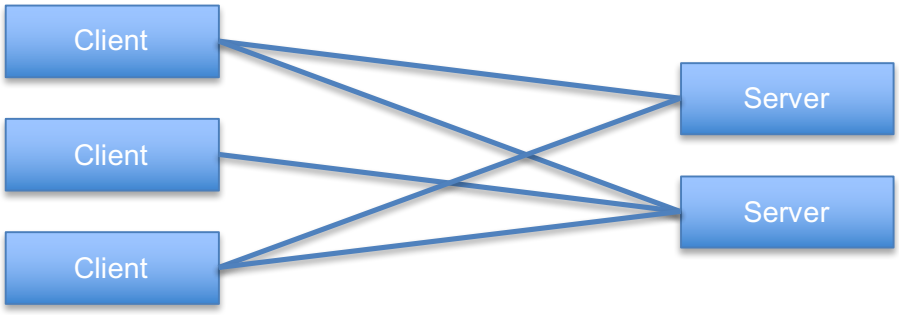

Figure 2. A typical EPICS system architecture.

For clients only reading data, an alternative to this architecture is possible with the introduction of a data aggregator. Clients then subscribe to a certain set of data directly from the aggregator, which also allows them to receive data that is not made available via EPICS, such as detector data, through a single interface. Using pvAccess and the pvData library, EPICS v4 can create and allow clients and servers to exchange complex or custom data. In this picture the aggregator would mainly be a broker or proxy for the data and metadata stream, and signal dynamic changes to those.

EPICS v4 is intended to be multi-platform, though it currently requires some modifications to run on Windows. The documentation in the website has a section with sample architectures, including service aggregation and data acquisition, and mentions multicast.

In terms of data streaming rate, tests at the Paul Scherrer Institut (PSI) showed it to be slightly faster than ZeroMQ, but at a higher data rate it appeared to lose packets. The throughput achieved streaming a set of events repeated by a multiplier factor is show in Figure 3; further tests at ISIS suggested that this appears to be due to CPU load. The SNS have tested it over 10 Gigabit Ethernet and have managed 100 million events per second, with the limit being CPU, not the network.

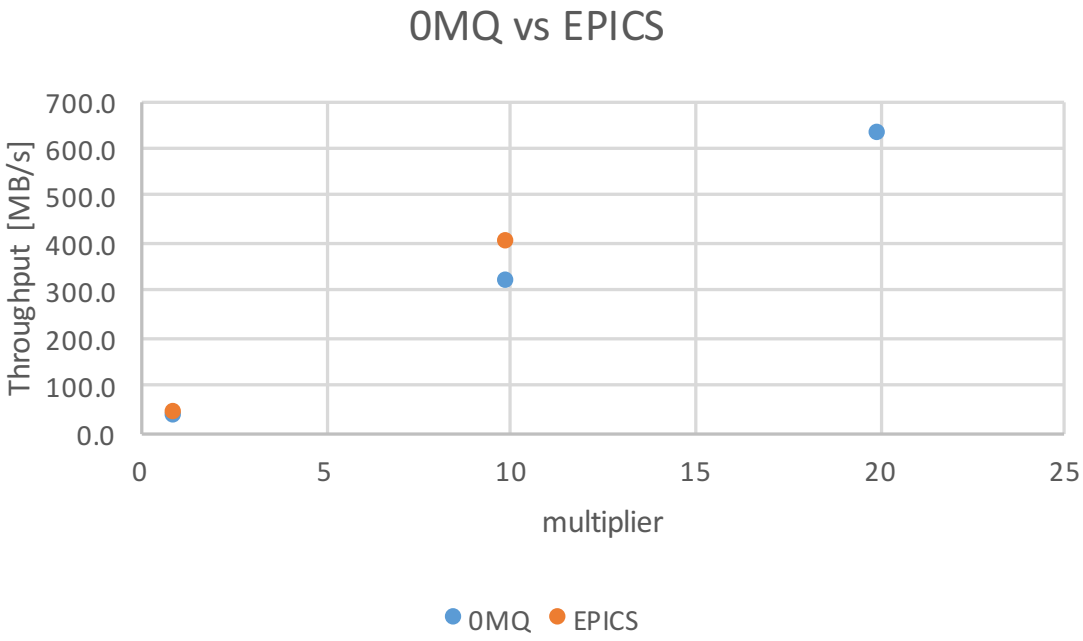

Figure 3. ZeroMQ and EPICS v4 throughput for repeated event data.

EPICS, and EPICS v4 in particular, are under active development and there is a roadmap. A number of facilities have investigated bits of it, but general uptake is slow, with most of them currently using EPICS 3. Currently, EPICS v4 depends on EPICS 3, and is backwards compatible with it.

On-line documentation is available on the EPICS website, including a getting started guide, developer guides, and there is Doxygen in-code documentation. Besides, a number of training courses are run regularly at EPICS meetings, with material available online. Examples are available with the source code packages and can be easily built using the provided Makefiles.

Servers can be implemented in C++ and Java, and clients in C++, Java and Python; as with EPICS 3, wrappers for other languages will most likely appear. Local support is available at ESS and the EPICS community support is excellent, with an active mailing list for questions and discussions. EPICS is used by a large community in the large science facilities, including NSLS-II for their control system, ESS, DLS and SNS. The Spallation Neutron Source (SNS) in Oak Ridge are using pvAccess for their area detector interfaces (ADnED).

Independently of whether EPICS be chosen as the streaming technology, the data aggregator software will have to interact with it at some level, as slow data will be provided via EPICS.

7.2.2.2 ZeroMQ

ZeroMQ is not a specialised solution for our domain; it is a high-performance asynchronous messaging library, available under a free open source LGPLv3 licence with a static linking exception. It supports not only a publish-subscribe networking methodology, but also implements other enterprise integrations patterns like push/pull, fan-out, etc. and is multi-platform.

The library does not provide tools for handling complex or custom data structures, but can easily be used with a number of existing alternatives for serialisation, such as JSON, BSON and Protocol Buffers. Different message delivery topologies can be implemented in a flexible way with the available patterns.

For the aggregator software, using ZeroMQ would require some kind of brokerage system to connect up streams. The aggregator would connect to each of the data sources using the appropriate protocols, obtain and serialise the data, and then make it available to subscribing clients using ZeroMQ.

Performance tests were carried out at PSI; in addition to the comparison with EPICS v4 (Figure 3), ZeroMQ throughput was evaluated against message size for different PSI instrument data sets (Figure 4) and against the number of clients (Figure 5).

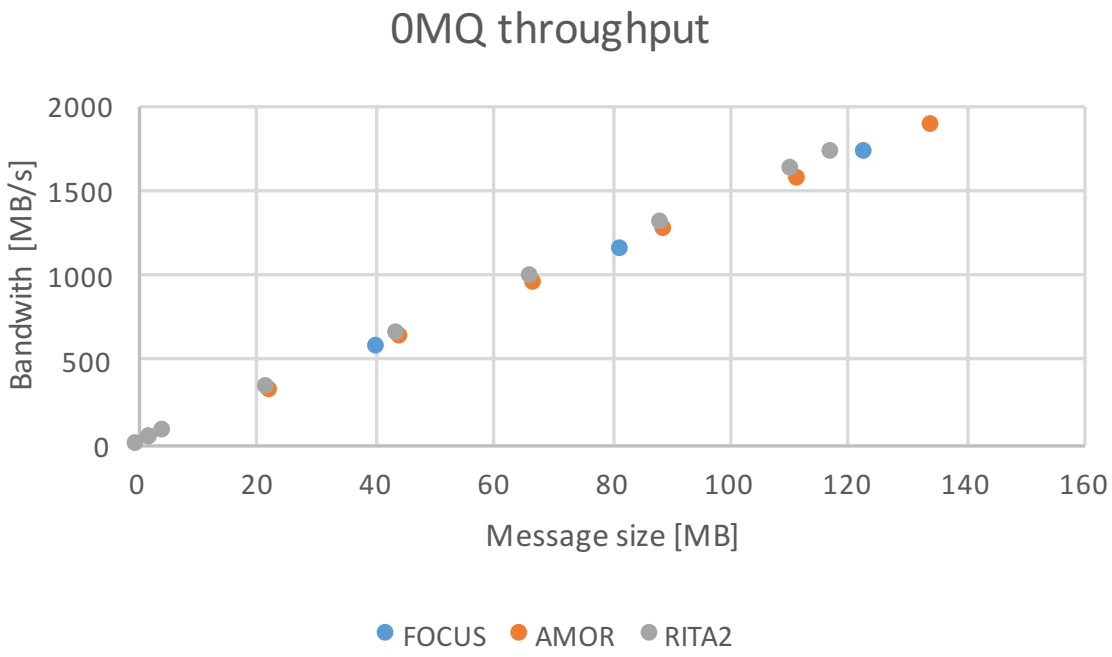

Figure 4. ZeroMQ throughput for different PSI instrument data sets and message sizes.

Figure 5. ZeroMQ throughput for different numbers of clients.

ZeroMQ is being actively developed, as the activity in its GitHub repository shows. The website states that ZeroMQ has no roadmap, but promises a stable API free with a sensible path for upgrades if required.

There is a large amount of documentation available at the ZeroMQ website and it is easy to create a basic example based on the on-line tutorials; besides, there is a book available from O'Reilly. The library has bindings for all mainstream languages and can be built from source code or installed from standard Linux distribution package repositories. The guide

available at the website contains examples for the different patterns, available for many of the supported languages.

ZeroMQ is widely used both in software developed at large scientific facilities and outside them. PSI have picked ZeroMQ over EPICS for large parts of their SwissFEL data acquisition and other X-ray facilities such as DESY and SLS also use it. Commercial support is available.

7.2.2.3 Apache Kafka

Apache Kafka is a publish-subscribe messaging system built as a distributed commit log. It is designed to be scalable and robust to hardware failures, with configurable data persistence and has very high throughput performance. It was originally developed at LinkedIn and is now an Apache Software Foundation open source project available under the Apache License 2.0.

Kafka works as a distributed publish-subscribe system with brokers and offers a higher level of abstraction than ZeroMQ. It is a scalable solution and adding and starting up more brokers in the cluster is simple. Compression is supported (gzip and snappy), as well as authentication and encryption (SSL and SASL). Kafka does not enforce a particular data format, but Apache Avro is a recommended option in the documentation; the same serialisation options as for ZeroMQ can be considered. It is implemented in JVM and used in systems dealing with huge quantities of data (e.g. at LinkedIn and Netflix), providing a guarantee that messages are delivered at least once. Kafka is based on a distributed log, so that, with proper replication configuration, machines in the cluster can go down without affecting service. For running the cluster, Unix-based operating systems like Linux and Solaris are recommended, and its dependencies are the JVM and Apache ZooKeeper. Current version is 0.9, but considered stable as evidenced by extensive use by large companies.

Being a publish-subscribe system, Kafka does not fetch the data from the sources by itself; instead, it relies on a publisher, called *producer* in Kafka terminology, to publish data as messages to a Kafka cluster, which is comprised of one or more servers referred to as *brokers*. Messages are published to a certain *topic* in the cluster and a client, or *consumer* in Kafka terminology, can then subscribe to the desired topics, receiving the data from the sources encompassed by them.

Error! Reference source not found. illustrates how Kafka could be used for aggregation, with sample systems from the ESS data acquisition architecture; boxes identified by *P* represent Kafka producers, while those identified by *C* represent consumers. Both producers and consumers can be implemented using the client libraries available for different programming languages.

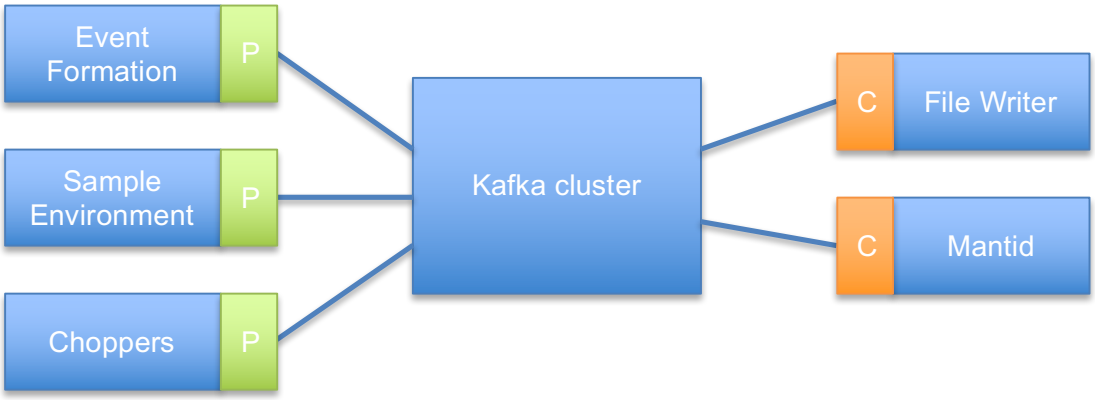

Figure 6. Data aggregation with Kafka in the ESS data acquisition architecture.

In a high level view, messages are written to topics in the Kafka cluster. A topic is identified by a name, used by producers when writing and by consumers when specifying the messages they want to subscribe to. The topic is kept in the cluster as a partitioned log, as shown in Figure 7**Error! Reference source not found.**.

Figure 7. Kafka topic and partitioned log.

Partitions are distributed over the different brokers in the cluster according to producer policies, in this way distributing the load among the servers, and topics can be replicated across them for fault tolerance. Kafka guarantees the time ordering of messages within the each partition, but not within the topic. Messages are written to disk and kept for a configurable period of time or maximum size and might be identified by a key. When using a key, there is an option for keeping a compacted log, in which only the most recent value for each key is kept; in this configuration, storage space taken by the log is reduced, while still keeping the possibility of replaying the data for reaching the last state of the system as defined by the keys and values.

The Kafka package comes with a tool for mirroring data between clusters, which can be use for increasing throughput and fault-tolerance. This could be use to mirror the data between clusters in Copenhagen and Lund, for example.

Apache Kafka is under active development and there is good online documentation for it and the majority of the client implementations for different languages, which include C++, Python and Java. The scripts and online documentation available make it easy to create a basic example. An O'Reilly book is about to be published.

Companies such as LinkedIn, Netflix, PayPal, and Spotify use Kafka. There is a large and growing community using it. Moreover, Kafka has a large ecosystem of tools, which may provide value for little of our time. For example, there is a consumer which processes messages and indexes them in elasticsearch, a management console for Kafka clusters, and easy integration with Storm, Spark and Hadoop. Commercial consulting and training is available.

7.2.3 Result and Conclusion

Apache Kafka was chosen as the technology for data aggregation and streaming. It implements much of the functionality required of the data aggregator, such as the publish-subscribe messaging pattern, and its clustered architecture allows the system to scale by adding more instances of the Kafka server. It is being used by a large community in projects that deal with very large amounts of data, is under active development, and has good documentation and commercial support available. In section 8, the use of Apache Kafka will be discussed further, presenting its main components and how they fit into the ESS data acquisition architecture according to the data aggregator software design.

ZeroMQ could have been the technology choice for the project. It is flexible, has very good performance, and is also used by a large community. It works on a lower level of abstraction when compared to Apache Kafka, however; choosing ZeroMQ would imply implementing much functionality that Kafka already provides.

EPICS v4 does not fit naturally with our requirements. Although slow data is going to be available via EPICS, aggregating heterogeneously structured data such as neutron events and metadata in a common channel for subscribers is not straightforward. Engineering an EPICS infrastructure and topology to fit our data transport needs would also move us away from the normal use case and we would not benefit from the community support in the same way as when deploying EPICS for controls purposes.

7.3 Technology Choice for Data Serialisation

The data aggregator software will aggregate and publish data that might come in various complex and custom data structures; these data will need serialisation to be sent across a network. The serialisation governs how information is laid out in files or network data packets, which often differs from how computer programs hold data in memory. There needs to be a common understanding of this scheme between the sender of data and the recipient. With the exception of EPICS v4, the technologies considered for data streaming do not provide a serialisation scheme. In this subsection we evaluate alternatives for this objective.

Alternatives can be evaluated against the criteria described in section 7.2, with some modifications. Relevant technical factors in this case are the presence of a schema, the language used to define it, and support to its evolution; potential advantages to use with a particular streaming technology; and stability. Performance can be evaluated by comparing encode and decode time and the size of the serialised messages.

In case it is needed, compression may be used to reduce the size of the messages to be sent across the network, at the expense of the added processing time for compressing and decompressing them.

The serialisation libraries and formats evaluated were Apache Avro, Apache Thrift, BSON, FlatBuffers and Protocol Buffers; for compression, Gzip and Snappy were considered. We first briefly present each of the serialisation alternatives, followed by the performance comparison.

Apache Avro is a data serialisation system available under the Apache License 2.0. It can use JSON or an interface description language (IDL) to define the schemata, with the IDL being more readable, and generates binary messages. Avro offers APIs in different languages, including C, C++ and Java; Avro C++ depends on some Boost libraries. There are relevant code examples and documentation and a large community uses it; besides,

there is a tool for managing Avro schemata in an Apache Kafka system, named Schema Registry.

Apache Thrift is a framework for cross-language services development available under the Apache License 2.0. It uses a simple and readable IDL for the schemata, generates binary messages and can work with many mainstream languages, including C++, Java and Python. Running a simple example is not difficult, but documentation and examples are focussed around the provision of services, which is not very relevant to our use. It is also used by a large community.

BSON, or Binary JSON, is a binary serialisation format that does not employ a predefined schema description, used as the primary data representation for MongoDB. There are drivers for the mainstream programming languages, including C++, Java, and Python The C++ mongo-cxx-driver is available under the Apache License 2.0, but is a new implementation and there is little documentation available other than basic examples. The BSON community is large, as it is used by MongoDB.

FlatBuffers is a cross-platform serialisation library, originally created at Google for performance-critical applications, available under the Apache License 2.0. It uses a simple IDL to describe schemata and represents complex data in a flat binary buffer. The library is available for the major programming languages, such as C++, Java and Python. The website provides good documentation with code examples.

Protocol Buffers is another Google library for serialising structured data, available under the Apache License 2.0. A simple and readable IDL is used for describing schemata and Languages such as C++, Java and Python are supported. There is good official documentation available and the user community is also large and can provide support and documentation. It is available in many common Linux distributions from the standard repository packages.

Figure 8 shows the results of the performance comparison of the serialisation libraries, with different compression alternatives. Neutron event data from an AMOR PSI instrument file was loaded and the time spent on encoding and decoding the data was recorded. FlatBuffers covers the widest spectrum of possibilities, resulting in the smallest message size when used with Gzip compression, and the smallest encode and decode time without compression. With Snappy compression, an intermediate result is obtained. For this reason, FlatBuffers was selected as the serialisation library for the data aggregator software.

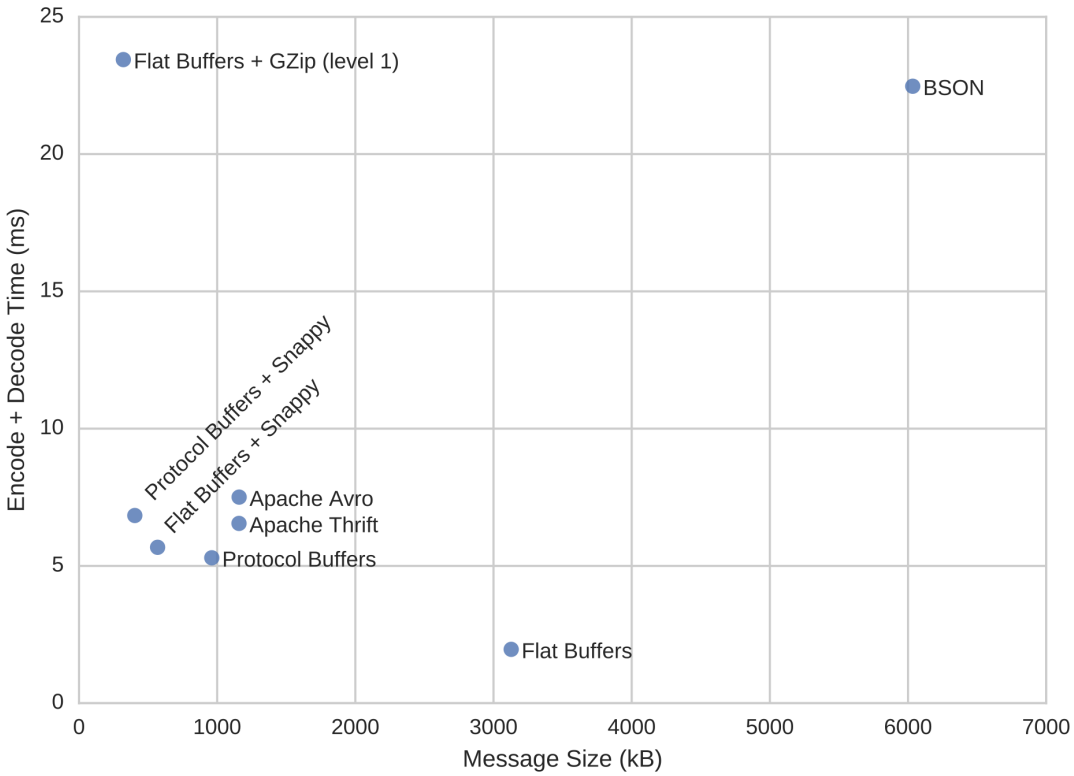

Figure 8. Comparison of serialisation and compression tools.

8 Roadmap – Development, Testing, and Deployment

8.1 System Design and Architecture

The design proposed for the data aggregation software is based on Apache Kafka as a distributed publish-subscribe messaging system, using Google FlatBuffers for data serialisation. As the Kafka broker is available as open source software, the work related to the Kafka cluster is mainly one of configuration and deployment. Development work is directed towards the Kafka producers and consumers, as data from the different sources need to be published to the cluster, and clients need a transparent way of obtaining them, without getting into the details of the Kafka architecture, topic structure, and serialisation. In sections 8.2 and 8.3 these components are discussed in more detail.

8.2 The Apache Kafka Cluster

An Apache Kafka cluster is composed of one or more brokers. The basic local configuration required for a broker to join a cluster is a unique number as a server identifier and the address of the Apache ZooKeeper servers that keep the distributed system configuration.

Apache ZooKeeper is used by the brokers in the cluster for storing the distributed configuration and many distributed computing issues such as leader election and replication are automatically managed by Kafka.

We expect the work with the cluster to consist mainly of planning, configuration and deployment. Planning involves determining the number of instances necessary to handle the streaming of data, according to the volume and rate of data in each instrument, as well as establishing a topic naming scheme, the partitioning strategy and replication factor, and choosing the adequate parameters for persistence periods or maximum log size. Configuration includes setting all the cluster parameters in accordance with planning and also tuning the servers after receiving feedback from tests and monitoring. As Apache Kafka is a distributed application comprised of many brokers running in different computers, deployment must also be carefully considered. Automation is therefore essential, as is the employment of efficient monitoring tools to evaluate the performance of the running system and be able to intervene before failures take place. There are open source tools available for monitoring and testing Kafka clusters, such as Kafka Manager and Kafka Monitor.

So far we have identified the data types that might be assigned to different topics for an instrument as detectors, metadata, choppers, monitors, images and control information. A naming scheme of “<beamline>_<datatype>” has been adopted for topic identification, with sample names being “HEIMDAL_monitors” and “C-SPEC_choppers”.

8.3 Data Producers and Consumers

Source code for data producers and consumers will make up the majority of the code developed in this task of the BrightnESS work package 5. Client libraries for Apache Kafka are available for this, including the C++ librdkafka and the Python Confluent Kafka.

The producers are responsible for obtaining the data from their original sources, which may be EPICS servers in case of sample environment and motion control, or the event processing system in case of neutron events. This data must then be published to the appropriate topics. It is a producer responsibility to select the partition attribution strategy, which defines how each message is attributed to each partition within a topic; a simple strategy is round-robin, but it can alternatively be done based on message keys.

Sending messages to the cluster can be done synchronously or asynchronously, and producers can be configured for parameters including the acknowledgement level expected from the cluster, the memory set for buffering messages, the batch size before messages to be written are sent to the cluster and the lingering time to wait for additional messages to be added to the batch before sending it.

Some EPICS variables will generate enough traffic to justify their own topic. Other variables change less frequently and it may be desirable to have them share a Kafka topic, in order to improve data locality and to keep the number of open files at a reasonable number without adding a large number of nodes in the cluster. This has also consequences for the Kafka message format; for a high-traffic variable with its own topic there is no need to send the name of the variable with each message. Also, it is likely that the schema does not change often, which allows the efficient use of FlatBuffers. On the other hand, the Kafka messages in topics that are shared by several EPICS variables must identify the variable; also, there may be the wish for more dynamic schemata. The possibility of multiple producers per topic makes it more challenging to attach any kind of state to a topic, e.g. schema.

Due to the existence of partitions, multiple clients can consume from the same topic with the guarantee that each message is consumed only once. This is achieved by adding the consumers to a *consumer group*. A consumer group is identified by a name that is simply

set in the client configuration. Consumers keep track of their current position in the partition using an offset, which is periodically committed to the Kafka cluster.

In the ESS data acquisition architecture, the software systems consuming the streamed data include both systems under the responsibility of the Data Management group, as is the case of the NeXus file writer, and under the responsibility of other groups, as live data reduction and visualisation. Taking this into consideration, the Kafka consumer can be offered as a library with an API that abstracts Kafka details away from clients. This will require interaction with the relevant stakeholders to elicit their requirements.

8.4 Serialisation and Schema Management

FlatBuffers was chosen as the serialisation library to be used by the Kafka producers and consumers. The FlatBuffers serialised data will be used internally by the data aggregator system, in a way that is transparent to the servers that provide the data being aggregated and the clients that consume the data. It is then a responsibility of the Kafka producers and consumers to perform the serialisation before messages are sent and deserialization when they are received.

The serialisation process relies on a schema defined in an IDL, which is then processed by the FlatBuffers compiler to generate the files in the chosen programming language, to be included by the producers and consumers. The complex data structures can then be serialised and deserialised by calling a simple function.

One important issue concerning serialisation is schema evolution. As the data aggregator and streamer is a distributed system, where information is sent across a network, both the producers and consumers must agree on the schema being used; as the system evolves, with bugs fixed and new functionality added, the schemata might need to be modified. This needs to be done in a way that does not break the existing producers and consumers. The Avro schema registry available with Apache Kafka is an example of how this problem can be addressed.

Schemata are currently being maintained in a common code repository under version control; changes to them can be tracked and reversed, in case it become necessary.

8.5 Collaboration and Tools

Planning and development coordination is being done using the Atlassian collaboration tools JIRA, Confluence and Bitbucket available at ESS. All project members have accounts that grant them access to the web-based tools, with the appropriate permissions for their tasks.

JIRA is a project and issue tracking tool focussed on agile projects. It allows teams to plan and follow progress using a ticket system, with customisable workflows. Inside a project, requirements, features, and tasks can be described in a ticket, which can then be assigned to users and to different stages in a workflow (e.g. selected, in progress, and blocked externally). The tickets can also contain comments and relationships to other tickets, establishing dependencies and breaking tasks into subtasks. A board view offers a quick overview of the current project activity, with the issues being addressed at the moment and the users responsible for them.

Confluence is a document collaboration system, where users can create and edit pages, with templates such as meeting notes and file lists available; pages can be organised in a

hierarchical structure. The Data Management group maintains a space for the data aggregation and streaming project, where documentation and discussions are kept. Confluence allows users to add comments to pages and specific sections of text, and also insert references to JIRA content.

Bitbucket is a product for source code repository hosting, supporting the Git and Mercurial distributed version control systems. Repositories accept pull requests, can be owned by teams and organised in projects, and might contain an issue tracker and a wiki. Repositories for the data aggregation and streaming software are owned by the Bitbucket team "europeanspallationsource" and are grouped under a project named Data Management.

Doxygen is a tool for code documentation. It enables publishing documentation in HTML and LaTeX format from comments maintained in the source code. Doxygen can be useful to document APIs provided by data aggregator software libraries.

Virtual meetings of the BrightnESS partners in Work Package 5 are held every two weeks to report progress and discuss open questions, as well as decide on the tasks to be addressed next. Tasks are entered into JIRA tickets, which are then assigned to a team member. Discussion and documentation exchange is carried out using Confluence, and development collaboration happens in the Bitbucket repositories, using Git as the version control system.

We adopt an agile approach to software development, with a focus on early delivery and testing. By having versions of the software available early, with partial functionality but in shape for deployment, integration testing can be carried out, facilitating interaction with stakeholders and allowing the discovery of problems. The feedback obtained from these activities can then be fed into the next iterations of the project, addressing requirements incrementally.

The source code is being developed under the BSD 2-clause licence. This is a simple to understand permissive open source licence. It allows anybody to use the software use (without warranty), distribute, or modifiy the software, on the condition that the licence and copyright notice is included. With that we hope we can maximise the interest and re-use of the project outcome at other facilities.

8.6 Testing and Deployment

Different types of test are being considered for the data aggregator software system. Unit tests can be added to each component we develop for the system, exercising and verifying the interfaces provided, with different tools used for different languages. Python, for example, comes with a unit testing package in its standard library; for C++, the Google Test library is open source software available under a BSD 3-clause licence and provides resources for running tests and mocking interfaces. The unit tests can be run in the development machine before code is committed to the repositories.

Code from the repositories can then be checked out, built, and have unit tests run on a build server. The Data Management group has a Jenkins instance available in the DMSC computing infrastructure; Jenkins is a build server that offers a web interface for projects, with a plugin system that includes components for publishing test results and sending alerts for build and test results.

Integration tests assemble the different software components of the project and verify they work appropriately together. Virtual machines are available in the DMSC computing

infrastructure for deploying Kafka brokers, producers and consumers for testing. For generating the data to be aggregated, simulators can be run, supplying data in different formats, as EPICS PVs from random numbers and motor simulators, and neutron events from NeXus files from operating neutron instruments at running facilities, such as PSI.

The Data Management group is currently deploying three test servers to the ESS integration laboratory in Lund. This laboratory is the central part of the ESS Instrument Integration Project (ESSIIP), and will be used to deploy and test the integration of neutron instrument hardware and software developed by different ESS groups, including the Chopper, Motion Control and Automation, Sample Environment, ICS and the DMSC. The three servers are going to be used to run software from the event formation and data aggregation tasks, integrating these systems to real equipment of the type that will later be deployed to the ESS instruments. In addition to these integration tasks they will also be used for performance test and evaluations within work package 5.

Tools for automated configuration management and application deployment are being evaluated. Ansible is an open source platform that addresses these needs, available under the GNU General Public Licence. Scripts with instructions to be carried out for specific hosts, called playbooks, can be run remotely via SSH.

For generating reproducible development environments, a Vagrant setup file is provided in a Data Management group Bitbucket repository. Vagrant is open source software available under the MIT licence and can create virtual machines from a setup file, starting from images called boxes, which contain an operating system and packages. After the virtual machine is created from the box, additional configuration steps can be carried out using shell scripts or configuration management systems like Ansible.

Docker is an open source tool available under the Apache License 2.0, providing software containerisation. It automates operating system-level virtualisation, allowing software to be packaged with all their dependencies inside a container, which can then be deployed to a Linux machine. Several containers can be run in the same machine and can be quickly started. Docker is being evaluated and used by other ESS groups, as the Instrument Data group and ICS, and could be used to standardise the deployment of component simulations used to generate data streams for testing.

9 Risks and Mitigation Strategies

There are many generic risks in any software project. In this section we will mainly focus on the specific risks to do with the technology choices made. Of the design decisions made, the choice of Apache Kafka as the technology for data streaming and aggregation carries the greatest implications. The resulting risks include not achieving the required data transport rates and latencies, and not achieving sufficient reliability in message delivery. Kafka has demonstrated its suitability for projects with large quantities of data and allows scaling with the addition of new brokers. To specifically validate its suitability for the data aggregation and streaming task at ESS, tests are planned to take place at the ESSIIP laboratory machines as soon as they as they become available. This will allow the team to evaluate the need to fallback to an alternative solution and do it as early as possible, therefore reducing rework.

The fallback solution for Apache Kafka is ZeroMQ; for its flexibility, ZeroMQ can be used to implement the functionality needed for the data aggregation and streaming, but with more

code to be developed and maintained by ESS, as it works at a lower level of abstraction. To reduce the amount of rework necessary in case this solution is necessary, good software engineering practices like abstracting away the Kafka client library in producers and consumers will be used; this would make switching to ZeroMQ easier.

The risk associated with the choice of FlatBuffers as the serialisation library is lower than the risk associated with Apache Kafka. Serialisation will be used internally by the system; thus it can be more easily changed, in case it cannot satisfy our requirements for performance and schema evolution. To reduce the amount of possible rework in case a change in the serialisation library is necessary, its use will also be abstracted away in code.

Setting up the Kafka cluster, which includes deploying and tuning the brokers, can be a demanding task because of the number of brokers and the distributed nature of the system; besides this, it is also necessary to monitor the cluster status and act to prevent and correct faults and failures. An associated risk is this task becoming excessively complex and time consuming for the number of people available in the project. To mitigate this risk, appropriate documentation will be produced for all stages in the deployment. We plan to provide tools for tuning and monitoring performance with as much automation as is reasonably feasible.

As the Data Aggregator Software project includes many different components to be developed, such as the different Kafka producers and consumers, we expect the associated code base to be extensive. This comes with the risk of technical debt, that is, the presence of code that solves problems in the short term, but becomes hard to maintain in a longer term. This risk is addressed by following good software engineering practices, including source code and configuration version control, testing, refactoring and code reviews when relevant. The recommended practices and policies are being documented and discussed using the collaboration tools.

10 Conclusion

The data aggregator software task in work package 5 is progressing as planned, with collaborators working together well. We decided to use Apache Kafka as the underlying technology for aggregation and streaming, and Google FlatBuffers as the serialisation library. Development work is going on with an agile approach focussed on early software tests and delivery, addressing the requirements in iterations. Collaboration tools are being used to track the progress of the project, discuss issues and maintain documentation, with virtual meetings held every two weeks.

Modules to generate simulated data streams for integration and performance evaluations are ready to be put into our testing and deployment infrastructure. This infrastructure consists of a build server and with virtual machines for deploying and running integration tests; as well as a physical lab space with three servers to be installed at the ESSIIP laboratory in Lund, where software from the Data Management group can be tested and integrated with real hardware and software from other ESS groups.

In the coming development cycles we are going to validate the technology choices and system design and architecture in both settings, addressing problems that might be discovered in the tests. The use of Kafka in projects handling large volumes of data is an indication that it can satisfy our requirements. Results from the tests and future project steps will be documented in JIRA, Confluence and Bitbucket.

11 List of Publications

- Akeroyd, F., *et al*. *A New Design for Live Neutron Event Data Visualisation for ISIS and ESS*. Poster accepted for NOBUGS 2016, Copenhagen, Denmark.
- Mukai, A., *et al*. *Development, testing and deployment of the ESS data aggregation and streaming software*. Poster accepted for NOBUGS 2016, Copenhagen, Denmark.
- Jones, *et al*. *ESS Event Data Streaming*. Oral contribution accepted for NOBUGS 2016, Copenhagen, Denmark.

12 References

- *ADnED*. (n.d.). Retrieved from https://github.com/areaDetector/ADnED
- *Ansible*. (n.d.). Retrieved from https://www.ansible.com
- *Apache Avro*. (n.d.). Retrieved from http://avro.apache.org
- *Apache Kafka*. (n.d.) Retrieved from http://kafka.apache.org
- *Apache Thrift*. (n.d.). Retrieved from http://thrift.apache.org
- *Atlassian*. (n.d.). Retrieved from https://www.atlassian.com
- *Benchmarking Apache Kafka: 2 Million Writes per Second (On Three Cheap Machines)*. (n.d.). Retrieved from https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines
- *BSD 2-clause "Simplified" License*. (n.d.). Retrieved from http://choosealicense.com/licenses/bsd-2-clause/
- *BSON*. (n.d.). Retrieved from http://bsonspec.org
- *ChannelFinder*. (n.d.). Retrieved from http://channelfinder.sourceforge.net
- *Confluent Kafka*. (n.d.). https://github.com/confluentinc/confluent-kafka-python
- *Data Management project repositories*. (n.d.). Retrieved from https://bitbucket.org/account/user/europeanspallationsource/projects/DM
- *Docker*. (n.d.). Retrieved from https://www.docker.com
- *Doxygen*. (n.d.). Retrieved from http://www.stack.nl/~dimitri/doxygen/
- Dworak, A., *et al*., *Middleware Trends and Market Leaders 2011*. 2011. In: Proceeding of ICALEPCS2011, Grenoble, France.
- *EPICS Version 4*. (n.d.). Retrieved from http://epics-pvdata.sourceforge.net/index.html
- *Experimental Physics and Industrial Control System*. (n.d.). Retrieved from http://www.aps.anl.gov/epics/
- *FlatBuffers*. (n.d.). Retrieved from http://google.github.io/flatbuffers/
- *Google Test* (n.d.). Retrieved from https://github.com/google/googletest
- Hensler, O., et al. *Controls Architecture for the Diagnostic Devices at the European XFEL*. 2012. In: Proceedings of PCaPAC2012, Kolkata, India.
- *Jenkins*. (n.d.). https://jenkins.io
- *JSON*. (n.d.). Retrieved from http://www.json.org
- *Kafka Manager*. (n.d.). Retrieved from https://github.com/yahoo/kafka-manager
- *Kafka Monitor*. (n.d.). Retrieved from https://github.com/linkedin/kafka-monitor
- Kasemir, K. U.; Guyotte, G. S.; Pearson, M. R. 2015. *EPICS V4 Evaluation for SNS Neutron Data*. In: Proceeding of ICALEPCS2015, Melbourne, Australia.
- *librdkafka*. (n.d.). Retrieved from https://github.com/edenhill/librdkafka
- *libzmq*. (n.d.). Retrieved from https://github.com/zeromq/libzmq

- *mongo-cxx-driver*. (n.d.). Retrieved from https://github.com/mongodb/mongo-cxx-driver
- *MongoDB*. (n.d.). Retrieved from https://www.mongodb.com
- *Neutron Detectors for ESS Instrument Projects*. (n.d.). Retrieved from https://ess-ics.atlassian.net/wiki/display/DG/Neutron+Detectors+for+ESS+Instrument+Projects
- *NeXus homepage*. (n.d.). Retrieved from http://www.nexusformat.org
- O. Arnold, *et al*., *Mantid—Data analysis and visualization package for neutron scattering and μSR experiments*, Nuclear Instruments and Methods in Physics Research Section A, Volume 764, 11 November 2014, Pages 156-166, http://dx.doi.org/10.1016/j.nima.2014.07.029.
- *Protocol Buffers*. (n.d.). Retrieved from https://developers.google.com/protocol-buffers/
- *The Process Variable Gateway*. (n.d.). Retrieved from http://www.aps.anl.gov/epics/extensions/gateway/index.php
- *Vagrant* (n.d.). Retrieved from https://www.vagrantup.com
- *ZeroMQ*. (n.d.) Retrieved from http://zeromq.org