Computing and Software at Belle II

David Dossett[1,*] for the Belle II Collaboration

[1]School of Physics (David Caro Building), The University of Melbourne, VIC 3010, Australia

Abstract. In 2019 the Belle II detector began collecting data from e^+e^- collisions at the SuperKEKB electron-positron collider. Belle II aims to collect a data sample of $50\,ab^{-1}$, 50 times larger than the previous generation of B-Factories. In order to process this data and release it to physics analysts there are many interconnected pieces of software and computing resources that must be utilised effectively. In this paper, the current Belle II computing systems, software, and data production activities are summarised.

1 Introduction

The results from the B-factories Belle [1] and BaBar [2] have confirmed the existence of large *CP* asymmetry in the b-quark system [3, 4] as predicted by the Cabibbo-Kobayashi-Maskawa (CKM) matrix [5]. However, the matter-antimatter asymmetry in the universe we live in cannot be explained by the theory alone. Therefore larger sources of *CP*-violating new physics (NP) should exist.

The Belle II experiment is the next-generation B-Factory experiment at the SuperKEKB electron-positron collider. In March 2019 the Belle II detector began collecting data, see Fig. 1 [6]. SuperKEKB is an asymmetric ($4\,GeV\ e^+$, $7\,GeV\ e^-$) collider at the High Energy Accelerator Research Organization (KEK) in Tsukuba, Japan. The full peak luminosity of $8 \times 10^{35}\,cm^{-2}\,s^{-1}$ is expected to be reached by 2026; Accumulating the planned integrated luminosity of $50\,ab^{-1}$ [7] by 2028. This huge increase in data will allow precision tests of the CKM matrix parameters, as well as in the τ sector.

2 Software

2.1 The Belle II software analysis framework

The Belle II Analysis Software Framework (`basf2`) [8] is a C++ and Python framework that provides common tools for manipulating Belle II data. `basf2` is used in all aspects of data processing at Belle II *e.g.* online data acquisition, track reconstruction, Monte-Carlo (MC) event generation, and for end-user analysis.

The ROOT analysis framework [9] is used for persistent storage of event data in a TTree structure; which is a columnar dataset where the independent columns are TBranches and the rows are integer labelled entries. At Belle II each recorded collision within an experiment and run, see Section 2.2, is assigned a unique event number and is stored as an entry in a TTree. Information relating to an event is stored in custom C++ classes attached to TBranches of a

*e-mail: david.dosssett@unimelb.edu.au

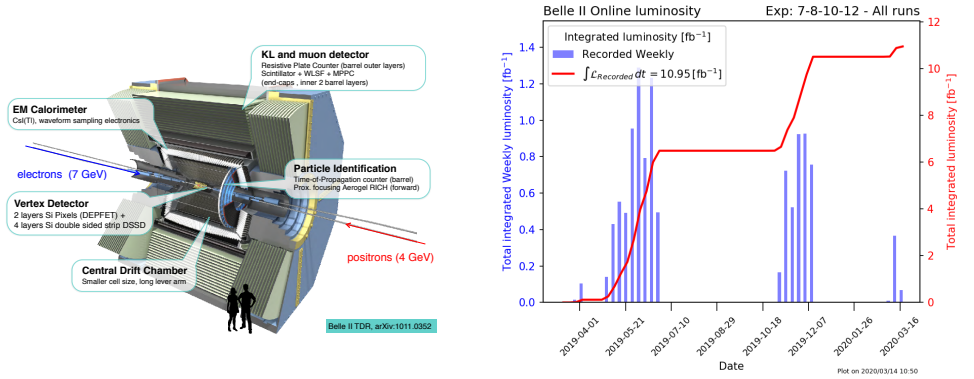

Figure 1. Left: A simple overview of the Belle II detector's main sub-detector components. **Right:** Recorded integrated luminosity measured from the online data acquisition system since 2019. Totals are given in weekly intervals and as the cumulative integrated luminosity.

TTree. Access to the current event's TBranches is globally available for all calling functions via an interface, the DataStore. `basf2` allows a user to define and configure algorithms, called *modules*, which are called once per event. Modules can be placed into a *path* object that runs the modules, in order, on each subsequent event. They may calculate new TBranch objects and save them with a unique name to the DataStore, ready to be used by a subsequent module in the path.

A Python API is also provided so that end-users can easily define the path of modules to be run in a declarative steering file. Many Python functions and classes have now been created to configure default paths for sections of Belle II processing *e.g.* track reconstruction and MC event generation. Additionally, since `basf2` can be run entirely from Python, Jupyter Notebook [10] integration tools have been developed. Allowing users and tutorials to run entire analyses inside an interactive notebook.

`basf2` also includes a conversion tool for older Belle data into the `basf2` data format [11]. These allow old Belle data to be directly compared against Belle II data using the same analysis tools. This provides validation of the analysis tools and improvements for analyses using the Belle dataset through more modern analysis techniques [12].

2.2 The conditions database interface

At Belle II the data taking periods are defined by *experiments* and *runs*, which are given incrementing integer numbers. An experiment is usually defined as a longer period of $\mathcal{O}$(month) with consistent detector conditions. It is usually incremented when a major detector/accelerator change occurred, or after a shutdown. The run number is incremented on demand from the Belle II Data Acquisition (DAQ) system and corresponds to $\leq$ 8 hours of data taking.

Conditions payloads are files, usually ROOT files containing a custom class, and are given a name and a unique payload id based on their checksum. End users do not need to configure their scripts for all the various required payloads in order to use an input file. Instead a `basf2` process only requires that the correct collection(s) of payloads should be specified in some way. These collections are defined by a unique name called a *global tag*, to which multiple payloads can be assigned. Payloads in a global tag are given an Interval of Validity (IoV), which defines the experiment and run numbers that they are valid for. A payload + IoV

combination is also given a revision number, so that if two of the same payload overlap in their IoV the higher revision will be selected. Each payload may be assigned to multiple IoVs under the same global tag [13].

Through this design, a `basf2` script needs to only specify a global tag name, or a list of global tags in order of priority. These are then queried when a payload type is requested by a `basf2` module, falling back through the global tags if no corresponding payload type for the current run number was found.

2.3 Calibration and alignment framework

In order to provide a common framework for calibration experts, the Calibration and Alignment Framework (CAF) was created in `basf2` [14]. The C++ components provide an interface to create, fill, and read ROOT data objects corresponding to each run as they are processed by `basf2`. This interface is achieved by providing a specialised base module class called a *collector module*. A calibration developer simply inherits from this class, allowing them to easily fill ROOT data objects labeled by the Belle II run number. Collectors are able to take advantage of `basf2`'s multiprocessing, if requested, by only allowing mergeable objects to be used and recombining them at the end of the process before final output.

Collector output files contain run-labeled data objects and the `basf2` processes creating them may be run in parallel over all data files. The calibration developer can then implement their specific algorithm code, which takes the collector output files and creates output payload classes. An *algorithm* base class is also provided to reduce code duplication. This class allows merging of data from specific runs in many collector output files, and provides helper functions for saving the payload classes as files for upload to the conditions database.

A Python API was created to configure combinations of collector modules and algorithms, while easily parallelizing job submission of collectors using different batch system backends, see Fig. 2. Dependent chains of collectors and algorithms can be made so that a calibration may derive payloads and pass them forwards for use in the next collector process. The Python CAF also allows a process to restart from certain checkpoint states once they have been safely reached. For example, if an algorithm step was badly configured and caused a crash after the collector step was finished, it is possible to change the Python script to correct the error and start from the algorithm step without having to redo the collector jobs again. Most data driven calibrations at Belle II now use the CAF, unless they require a difficult to automate analysis.

3 Collaborative tools

Management of software development with a worldwide collaboration can be a difficult task. Tools such as software version control, issue tracking, and wikis are all useful for maintaining software quality and communicating between developers. At Belle II we use a large array of commercial and custom tools.

In 2016 replacement of the KEK Computing Centre (KEKCC) infrastructure raised the issue of consolidating the various Belle II collaborative tools [15]. It was desired that existing tools should be migrated to a well operated computing centre outside of KEK. The tools were migrated to the Deutsches Elektronen-Synchrotron laboratory (DESY) in Germany. Centralising the tools allowed for user authentication and authorisation from the DESY LDAP server, allowing one set of user credentials to be used for all of them.

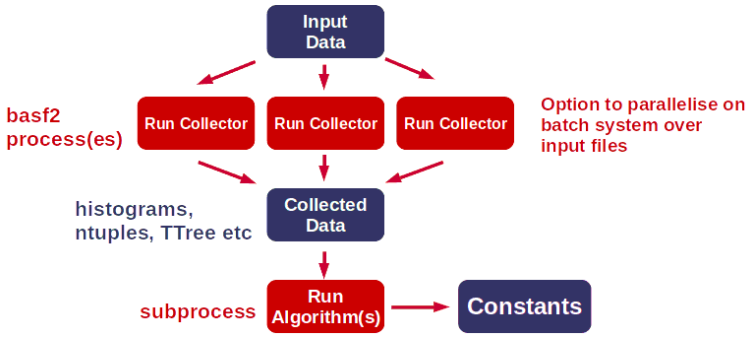

Figure 2. Data processing flow for a single iteration of a single calibration in the CAF. The outputs of the workflow are calibration constants saved to files. Most calibrations will iterate this process, using the previous constants as input, and may chain dependent calibrations together.

Table 1. Some examples of collaborative services hosted by DESY. The first four are bundled as part of the Atlassian tools [16]

Tool	Description
Bitbucket	Git repository hosting, as well as pull request and release handling
Confluence	Collaborative wiki
Jira	Generic task and issue tracker
Bamboo	Software build and continuous integration management
Indico [17]	Event management system
Invenio [18]	Document and article repository
Askbot [19]	Question and answer forum
Sympa [20]	Mailing lists
Sphinx [21]	`basf2` Python API documentation
Run Registry	A custom Django [22] web app to store and update information about run conditions and quality flags

4 Data processing

4.1 Data model

As with all large data HEP experiments, a balance must be struck between the cost of computing storage and available CPU resources. Storing objects that are costly (in CPU time) to produce in specific file formats can allow CPU resources to be saved and for user analysis jobs to complete more quickly. Also, the filtering or *skimming* of events containing different particle combinations into separate files allows every analysis using those combinations to avoid costly reconstruction time on events that their selection would reject anyway. For a high luminosity experiment analysing rare decays this is a very beneficial strategy.

At Belle II we use a relatively small number of file formats, see Table 2. uDST files are provided on the the only ones that a physics analysis will ever use as input, usually outputting an N-tuple for local analysis. The flow of data from detector (or MC production job) to end user is shown in Fig. 3.

Table 2. Data file formats used in the Belle II data model. There are no specific MC file formats since `basf2` can perform all the necessary steps in one process to generate mDST files with the extra MC truth objects included.

File Format	Description
raw (SROOT)	Custom Sequential ROOT file for raw detector data. This is the format written out by the DAQ system. It is converted to the smaller ROOT format and then deleted once validation of the conversion is complete.
raw (ROOT)	Standard ROOT file for raw detector data after conversion from SROOT.
mDST	Contains reconstructed tracks, clusters, and particle identification (PID) objects.
uDST	Contains everything in mDST but also some particle combinations useful for user analysis processes. These files are produced during skimming processes for analysis event types. User analyses can only use these files, not mDST.
cDST	A special format for calibration processes. Similar in content to mDST but also including raw objects. These files are temporary, currently not uploaded to remote storage, and are deleted when no longer useful for calibration studies.

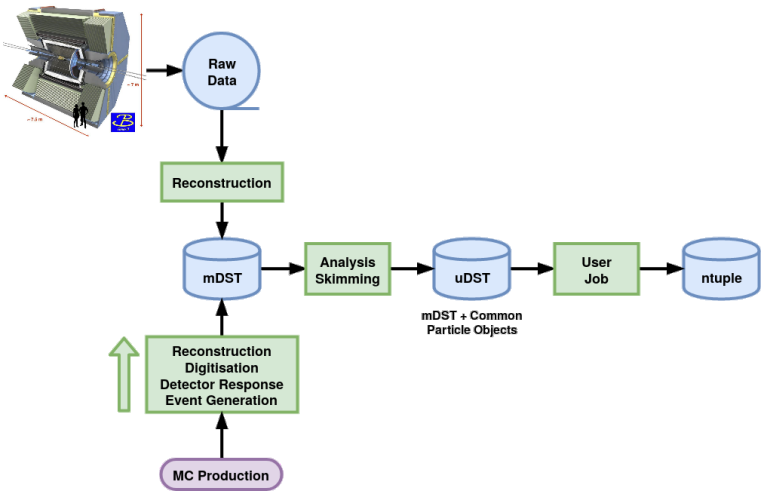

Figure 3. Simplified data model of the Belle II experiment.

4.2 Data handling at KEK

As shown in Fig. 3, the first step for a collision event is to be written out by the Belle II DAQ system. The High Level Trigger (HLT) uses `basf2` to perform essentially the same reconstruction as offline reconstruction. Calibration event flags are recorded by the HLT but not used for selection of events. HLT selection cuts are made to filter the recorded events before writing them out in SROOT format. Reconstructed objects are thrown away, but an object to store the various trigger decisions/flags is stored so that calibration events can be quickly filtered out for prompt processing at a later stage.

After the DAQ marks a run as closed the files must be copied out of the DAQ network so that they can be registered and used for processing. The flow of data and the

SROOT → ROOT conversion is shown in Fig. 4. At every step checksums are created and tested, as well as event counts being checked against the DAQ database values. Recent development for this copy and conversion procedure has produced a set of automation and monitoring tools. A Django monitoring dashboard has been created, along with a REST API [23] to query for the runs that have been successfully completed.

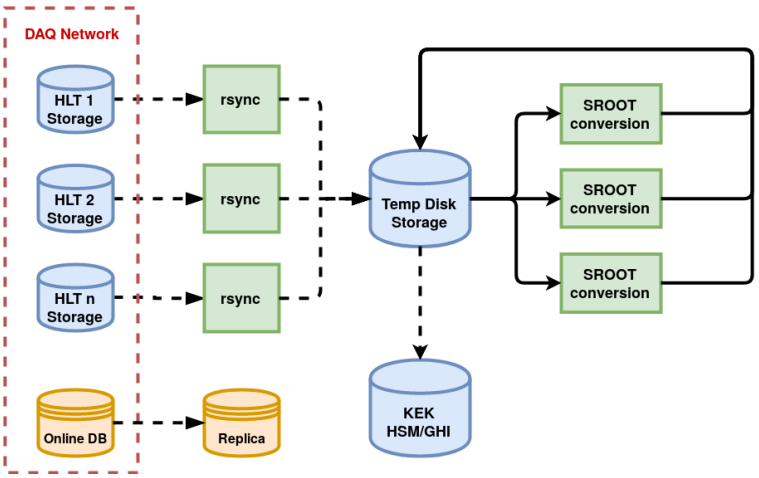

Figure 4. Flow of raw data files after a run is closed by the DAQ. SROOT files must first be copied out of the DAQ network. Then they are converted to ROOT format and stored at KEKCC, ready for upload to remote data centre storage and prompt processing.

Once these processes are complete the raw data can be used for calibration studies at KEKCC and copied to remote data centre storage for safety.

4.3 Prompt processing

One of the first processes run on raw data is the extraction of events useful to calibration studies. These are *HLT skims* which do not perform reconstruction, but filter and copy out events that passed the relevant HLT calibration flags. The resulting skimmed files are stored on disk at KEKCC temporarily, ready for calibration processes to use.

Before mDST production can begin, the runs contained in the production must be calibrated. Belle II has two mDST production schemes, reprocessing and prompt. Reprocessing mDST files occurs approximately every 6 months and uses the best possible calibration payloads available. These mDST files are usable for published analyses. Prompt mDSTs can be produced using prompt calibration payloads which should be as good as possible, but not usable for a final published paper, see Fig. 5. However the prompt data should be released within a few weeks of the data being taken and can be used as a starting point for analyses or for detector performance studies.

The prompt calibration process has many steps and is prone to human error causing delays. To reduce manual calibration tasks Belle II has created an automatic prompt calibration server based on the workflow management software Airflow [24] and the Flask web framework [25]. It allows a prompt calibration to be defined via a web interface, and for calibration experts to submit their previously tested prompt CAF scripts on automatically collected calibration data files at KEKCC. For a detailed description of the work, see [26].

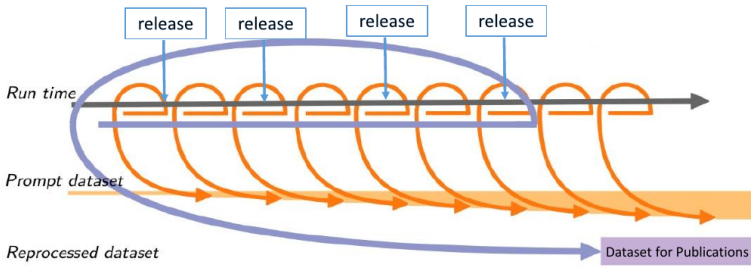

Figure 5. The rough scheme of prompt processing vs. major reprocessing. Releases are `basf2` software versions, the prompt dataset will contain data reprocessed from several `basf2` versions. A new reprocessed dataset is made approximately every 6 months using old + new data and the best calibration constants available.

4.4 Distributed computing

The Belle II computing systems must manage processing of ~60PB of raw data files, plus store a copy for safety at sites outside Japan. Producing MC data and reconstructing mDST/uDST files also contribute comparable amounts of storage space, while also dominating the CPU requirements. Belle II has more than 800 physics research members spread over 26 countries around the world. It is therefore natural that Belle II has adopted a distributed computing (grid) model based on existing technologies [27] in order to provide analysts access to files and to take advantage of resources wherever possible, see Fig. 6.

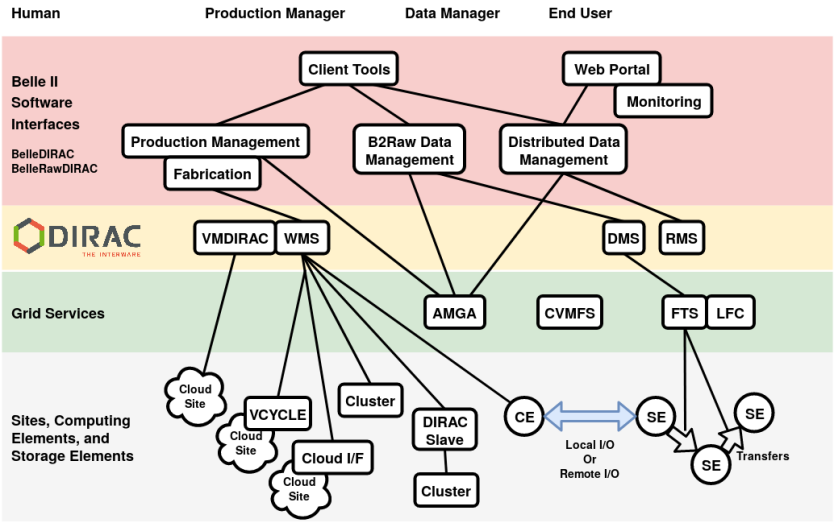

Figure 6. Distributed computing components used at the Belle II experiment.

DIRAC [28] was chosen as the workload and data management system to interface with the heterogeneous resources of the Belle II computing sites. AMGA [29] was chosen as a metadata service and we use LFC (LCG File Catalog) [30] as the file replica catalog.

In order to add Belle II specific production functionality to DIRAC, a custom extension has been developed called BelleDIRAC [31]. The Belle II MC grid production system is contained within this extension. Additionally a Distributed Data Management system (DDM) has been developed. It uses the DIRAC RMS and adds functionality for bulk transfers, a simple data distribution model, and performance monitoring, see Fig. 7. However, investigations to replace the custom DDM with Rucio [32] are now underway in order to leverage this promising tool.

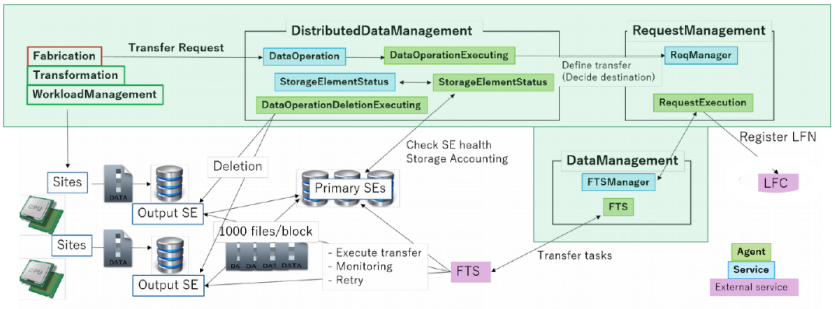

Figure 7. Distributed data management DIRAC components and their connections to the FTS, storage elements, and LFC.

Before raw data can be processed, the files must be uploaded to grid accessible storage and registered in the file and metadata catalogs. Another DIRAC extension, BelleRawDIRAC, has been created for this purpose. This uses a set of agents and services to perform the extraction of metadata from files, initial upload to grid storage, and replication to raw data centres. See Fig. 8 for a summary and [33] for a detailed description.

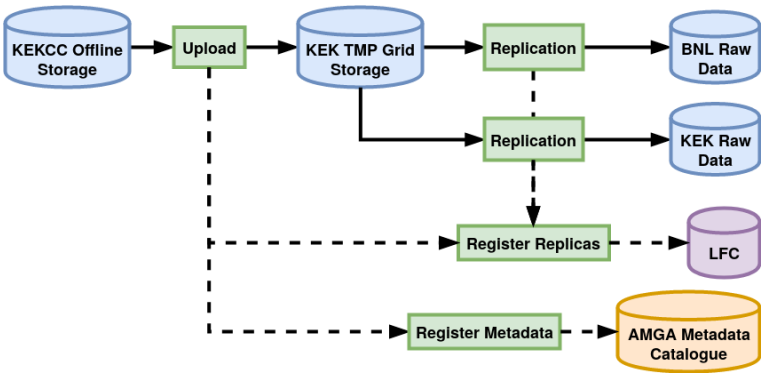

Figure 8. Simplified diagram of the processes completed by BelleRawDIRAC.

5 Summary

Belle II has begun taking data and is now releasing first results from the initial datasets. Software development contributions and knowledge transfer has been made easier through the use of collaborative tools. New development work in the past year has been put towards

making the initial data processing as automated as possible. We will be attempting to identify problem areas and reduce manual intervention throughout the coming year. Investigations into leveraging new distributed computing tools like Rucio is now a priority for the distributed computing group.

References

[1] The Belle Collaboration, *The Belle Detector*, Nucl. Instrum. Meth. A **479**, 117-232 (2002)

[2] The BaBar Collaboration, *The BABAR Detector*, Nucl. Instrum. Meth. A **479**, 1-116 (2002)

[3] The Belle Collaboration, *Measurement of the CP violation parameter $sin2\phi_1$ in B^0_d meson decays*, Phys. Rev. Lett. **86**, 2509-2514 (2001)

[4] The BaBar Collaboration, *Measurement of CP violating asymmetries in B^0 decays to CP eigenstates*, Phys. Rev. Lett. **86**, 2515-2522 (2001)

[5] Makoto Kobayshi, and Toshihide Maskawa, *CP Violation in the Renormalizable Theory of Weak Interaction*, Prog. Theor. Phys., **49**, 652-657 (1973)

[6] Belle II Collaboration, *Measurement of the integrated luminosity of the Phase 2 data of the Belle II experiment*, arXiv:1910.05365 [hep-ex] (2019)

[7] Tetsuo Abe *et al.*, *Belle II Technical Design Report*, arXiv:1011.0352 [physics.ins-det] (2019)

[8] Thomas Kuhr *et al.*, *The Belle II Core Software*, Comput.Softw.Big Sci. **3** 1 (2019)

[9] Rene Brun and Fons Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Nucl. Inst. Meth. A **389** 81 (1997)

[10] Jupyter Project, https://jupyter.org/

[11] Moritz Gelb *et al.*, *B2BII - Data conversion from Belle to Belle II*, arXiv:1810.00019 [hep-ex] (2018)

[12] Marcus Prim *et al.*, *Search for $B^+ \to \mu^+ \nu_\mu$ and $B^+ \to \mu^+ N$ with inclusive tagging*, Phys. Rev. D, **101** 3 (2020)

[13] Lynn Wood, Marko Bracko, Todd Elsethagen, Kevin Fox, Carlos Gamboa, Thomas Kuhr, Martin Ritter, *Performance of the Belle II Conditions Database*, EPJ Web of Conferences **214**, (2019)

[14] David Dossett *et al.*, *Status of the calibration and alignment framework at the Belle II experiment*, J. Phys. Conf. Ser. **898** (2017)

[15] Nils Braun *et al.*, *Migrating the Belle II collaborative services and tools*, J. Phys. Conf. Ser. **898** 102014 (2017)

[16] Atlassian tool suite, https://www.atlassian.com

[17] Indico, http://indico-software.org

[18] Invenio Digital Library Framework, https://invenio-software.org

[19] Askbot, https://askbot.org

[20] Sympa mailing list server, https://www.sympa.org

[21] Sphinx Python documentation generator, https://www.sphinx-doc.org

[22] Django web framework, https://www.djangoproject.com

[23] Roy Thomas Fielding, *Architectural Styles and the Design of Network-based Software Architectures.*, Doctoral dissertation, University of California, Irvine (2000)

[24] Apache Airflow, https://airflow.apache.org/

[25] Flask, https://flask.palletsprojects.com

[26] David Dossett, and Martin Sevior, *Prompt calibration automation at Belle II*, also in this issue

[27] Vikas Bansal, *GRID Computing at Belle II*, arXiv:1511.06760 [physics.comp-ph] (2015)

[28] Federico Stagni *et al.*, *DIRACGrid/DIRAC: v6r20p15*, https://doi.org/10.5281/zenodo.1451647 (2018)

[29] Sungsik Ahn *et al.*, *Design of the Advanced Metadata Service System with AMGA for the Belle II Experiment*, Journal of the Korean Physics Society **57** issue 4, 715-724 (2010)

[30] *LCG File Catalog*, https://twiki.cern.ch/twiki/bin/view/LCG/WebHome

[31] Hideki Miyake *et al.*, *Belle II production system*, J. Phys. Conf. Ser. **664**, no. 5, 052028 (2015)

[32] Martin Barisits *et al.*, *Rucio: Scientific Data Management*, Comput Softw Big Sci **3** 11 (2019)

[33] Michel Hernández Villanueva, and Ikuo Ueda, *The Belle II Raw Data Management System*, also in this issue