Performance of the Belle II Conditions Database

Lynn Wood[1,*], *Marko* Bracko[2], *Todd* Elsethagen[1], *Kevin* Fox[1], *Carlos Fernando* Gamboa[3], *Thomas* Kuhr[4], and *Martin* Ritter[4]

[1]Pacific Northwest National Laboratory, Richland, Washington, USA
[2]Institut Jožef Stefan, Ljubljana, Slovenia
[3]Brookhaven National Laboratory, Upton, New York, USA
[4]Ludwig-Maximilians-Universität, München, Germany

Abstract. The Belle II experiment at KEK observed its first collisions in the summer of 2018. Processing the large amounts of data that will be produced requires conditions data to be readily available to systems worldwide in a fast and efficient manner that is straightforward for both the user and maintainer. This was accomplished by relying on industry-standard tools and methods: the conditions database is built as an HTTP REST service using tools such as Swagger for the API interface development, Payara for the Java EE application server, and Squid for the caching proxy. This article presents the design of the Belle II conditions database production environment as well as details about the capabilities and performance during both Monte Carlo campaigns and data reprocessing.

1 The Belle II Experiment

The Belle II experiment is part of a broad-based search for new physics in the Intensity Frontier, focused on precisely measuring and comparing with theory branching fractions, angular distributions, CP asymmetries, forward-backward asymmetries, and a host of other observables. The SuperKEKB accelerator upgrade will provide 40x the instantaneous luminosity of KEKB and 50x the data taken with Belle. Initial "Phase 2" data-taking started in May 2018, concurrent with beam-tuning efforts, and "Phase 3" full running starts in 2019.

Figure 1 presents the various interactions that the conditions database needs to support. The primary purpose of the conditions database is to support data processing across the international Belle II computing grid, shown as the WAN connection at lower left that connects to the KEK computing center and Tier 1 computing centers, as well as interactive users. Data enters the database via the calibration cycle, which uses data collected by the Belle II DAQ at KEK and processes it into the appropriate format for the conditions database. Currently this is done offline, but is envisioned to become significantly more automated as Belle II matures.

[*] Corresponding author: lynn.wood@pnnl.gov

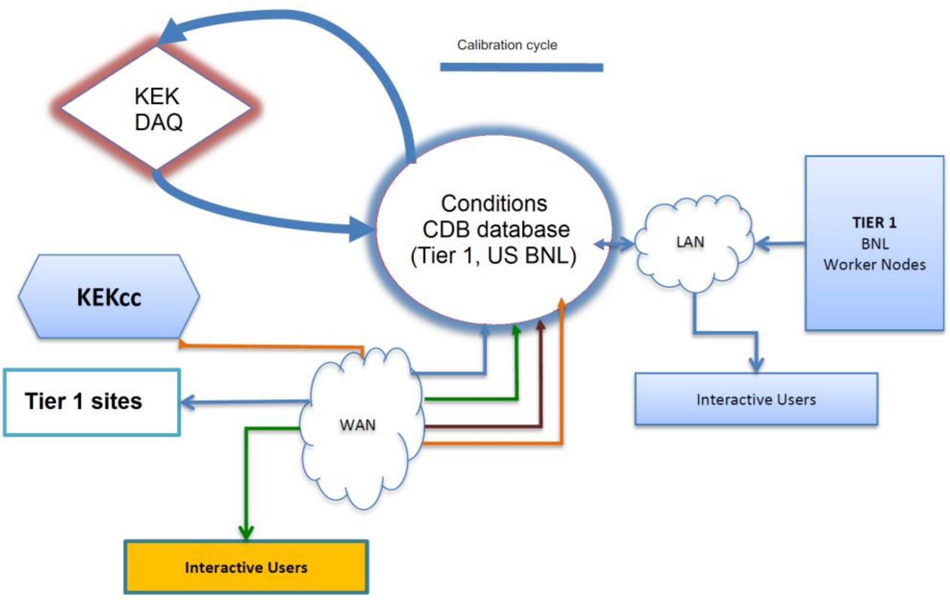


Fig. 1. Interactions of the Belle II conditions database with production nodes, interactive users, and the Belle II data acquisition system.

2 Belle II Database Software Architecture

2.1 Concepts

A conditions database holds time-dependent status of detectors for data processing and reprocessing, consisting of either constants or time-varying parameters such as calibration settings, geometry, alignment, etc.

User access to the Belle II database is through a REST API, meaning that all accesses take the form of HTTP requests. For example, the following HTTP request would return a list of payloads for a given global tag, experiment and run number:

iovPayloads?gtName=production&expNumber=3&runNumber=345

This type of interface provides several key advantages to Belle II: it restricts DB access to allowed operations only, and makes it easy to scale with industry standard (HTTP) tools. Responses can be formatted in either XML or JSON format.

Payloads are kept as external files outside of the database; a request for a payload returns a partial URL of payload file, which can be combined with a hostname to access a file server containing the payload. This method avoids a database bottleneck for large payload files, and allows for different scenarios for file storage and transfer.

2.2 Global Tags and Intervals of Validity

The data in the Belle II conditions database is organized using several structures. Intervals of Validity (IOVs) specify the starting and ending experiments and runs for which a payload is valid. A given payload may have multiple IOVs assigned to it if it is valid for different periods of time.

A Global Tag contains a list of IOV-payload relationships, and is used to select a complete set of conditions for a given reprocessing effort. The IOVs and payloads can be reused in different global tags, and the global tags can be classified as different types (development, release, online, data, etc.) and have different status over time (new, published, invalid, etc.).

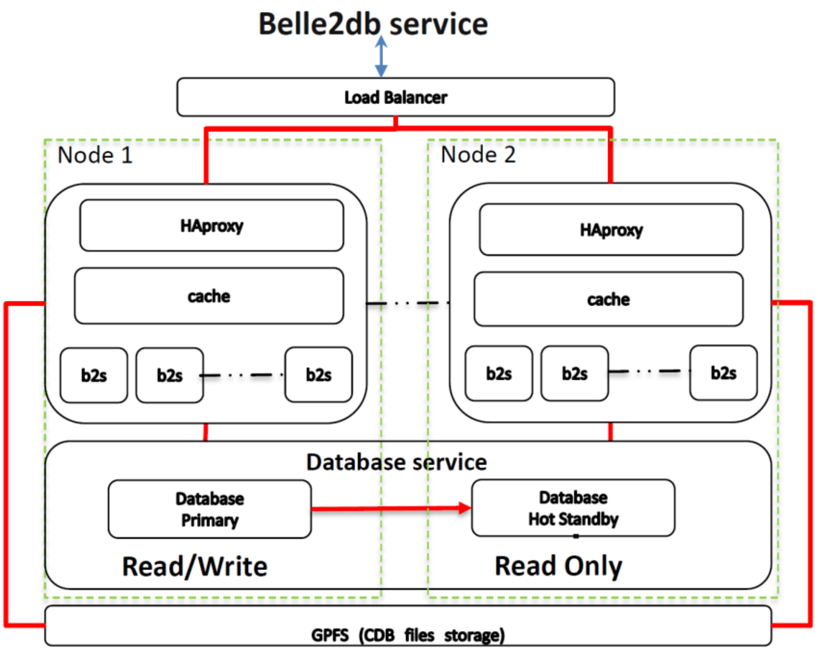
2.3 Service Layers

When operating at the expected full processing capacity, the Belle II computing grid will generate ~100 database requests per second, and can burst to significantly higher rates. To attain the required performance, multiple levels by relying on commercial tools designed for scalability are used.

Squid HTTP caches [1] are used to support repeated requests for the same query. These are configured as reverse proxy, i.e. optimized for many clients and few servers. These serve to cache the most common global tags in Belle II. Requests that are not in the Squid caches reach the “b2s” Belle II service layer. This layer is implemented in Java using Payara Micro [2], a microservice JavaEE server, and translates REST requests into SQL queries. The REST API is built using Swagger [3], which can auto-generate client and server code, as well as provide interactive online documentation of the API. Below these layers is the PostgreSQL [4] relational database, which resides on the IBM General Parallel File System (GPFS) [5].

3 Belle II Database Server Architecture

As mentioned previously, the database and payloads are kept on separate servers for improved performance and support. Both servers are built with multiple instances of all components, using Docker [6] containers to provide easy support, greater reliability, and improved performance. As shown on the top in Figure 2, the database service consists of multiple nodes behind an HTTP load balancer. Each node has its own dedicated Squid cache and multiple instances of the b2s service, and connect to the single database service. The database service contains both a primary database as well as a hot standby backup database to minimize any downtime.



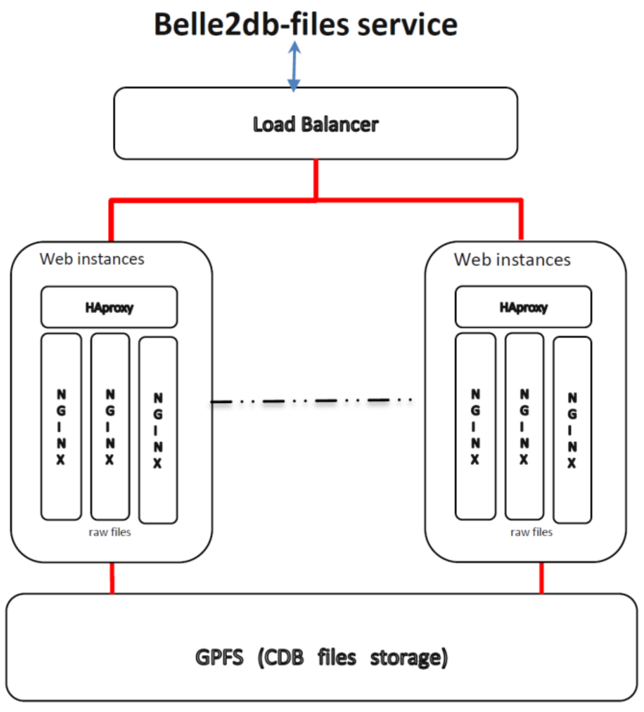


Fig. 2. Belle II database server architecture: (top) database server, (bottom) payload file server.

As shown at the bottom in Figure 2, the payload file is designed similarly, with multiple nodes behind an HTTP load balancer. Each node contains multiple instances of the NGINX [7] high-performance HTTP server, and the payload files themselves are stored in the IBM GPFS accessible by all nodes.

4 Client Interfaces

The primary interface is through the Belle II Analysis Software Framework [8], known as basf2. To simplify efforts, several assumptions are made about the conditions data. First, payload IOV granularity is assumed to be at the run level, as opposed to time or event granularity. This is expected to be sufficient for the majority of the conditions data in Belle II, but if sub-run granularity is required basf2 provides support for storing event number-based granularity dependence inside of a single run payload. Another assumption made is that all payloads are stored as ROOT [9] objects. This is not a requirement of the database system itself since the payloads are simply files stored on a file server, but makes the analysis software more manageable.

The DBStore service inside of basf2 manages storage and automatic updates of payloads as data is processed. This is handled through C++ template classes that always provide the user with a pointer to the currently-correct payload. The DBStore service ensures that the pointer is updated when a new IOV becomes valid for a payload, makes this run-dependence transparent to the user.

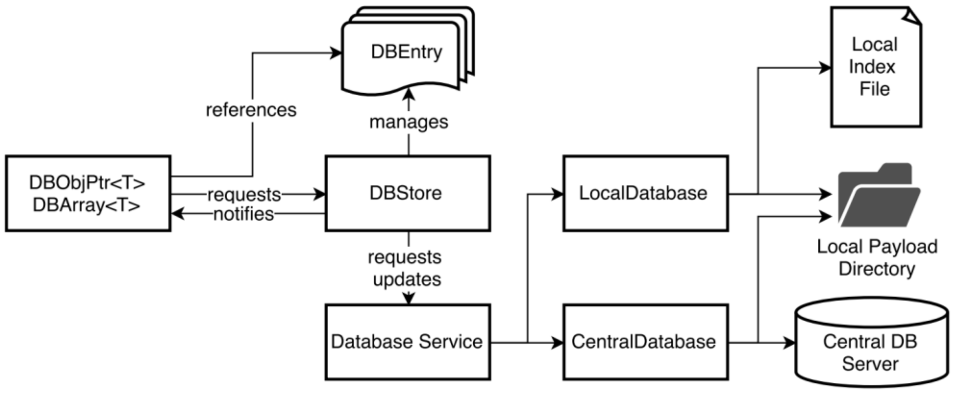


Fig. 3. Relationships between objects and modules in the basf2 analysis software framework. The user only needs to interact with the DBObjPtr and DBArray classes at left.

basf2 also supports accessing IOVs and payloads from either a remote database or a local directory, allowing development and test of new conditions data payloads before loading them into the global database. This is handled by creating a “chain” of database locations to search for a given payload in the basf2 steering file. The relationship between the conditions database components in basf2 is shown in Figure 3.

Due to the easy accessibility of the HTTP API, several other interfaces are also available. The Swagger tool used to develop the API provides online documentation of all queries, and allows the user to interactively send queries and view responses. A command-line client written in Python [10] is also available, and is currently the primary method of adding new payloads to the database.

5 Data Monitoring and Performance

5.1 Service Layers

The Belle II conditions database is a sophisticated system with many interactions between the different service layers. Extensive monitoring of all layers is implemented using Prometheus [11] and Grafana [12]; example parameters to be monitored include CPU and RAM usage, disk latency, input HTTP request rate, HTTP response types, and so on.

5.2 Database Performance

The current Monte Carlo campaigns and the data reprocessing for data taken in 2018 do not put a significant load on the DB and payload file servers. The current biggest stressor is users submitting large numbers of non-grid-based jobs, which cause a significant spike in requests over a short period of time. To ensure that the database is performing to the required level, directed tests are necessary to confirm that performance is sufficient for the expected full processing load.

Multiple types of tests are performed to ensure proper database behavior via Monte Carlo job submissions to the grid, scripting, and the Gatling HTTP load tester [13]. These tests include verification of basf2 software releases as well as specific scripts to generate “problem” workflows, such as the non-grid user submissions mentioned above.

In one example of performance testing, the Gatling tool was configured to generate 400 requests per second for a period of 15 minutes. The requests were randomly selected from a list of 21,000 actual request types recorded from HTTP logs. As the test ran, the network bandwidth and CPU usage was monitored, as well as the actual rates of requests submitted and responses received. No performance issues were seen during this test, and all parameters showed steady values, indicating sufficient resources were available for this type of workload.

6 Upcoming Development

While already in use by the Belle II experiment, a number of improvements are planned and in progress for the conditions database.

Authentication and authorization is currently not present for database access. The expectation is that all reads will be open, but write access needs to be user authenticated and authorized. Several methods of implementing this functionality are being investigated, including leveraging the existing X.509 certification authentication used for Belle II grid processing. It has been proposed to support three roles for database users: “user”, which would allow read-only access; “developer”, which would support adding data to existing global tags; and “coordinator”, who can create and manipulate global tags. These access rights would also vary with the type of global tag, to better control the update process.

While the implemented servers are fully capable of supporting Belle II during full production, the loss of networking access to the single location at Brookhaven National Laboratory would prevent proper operation of data reprocessing. To this end, replication of the system to other sites is being investigated. An intermediate solution being used currently is replicating the database contents and payload files on a distributed file system (CVMFS [14]), but as the database contents grow that may become a burden on the distributed grid site nodes. PostgreSQL supports standard replication such as streaming or log-based methods, although replication to remote sites is more complicated. Because of this, persistent remote Squid caching is being considered as a solution as well, where long cache invalidation times are implemented.

7 Conclusions

The Belle II Conditions Database was designed using industry-standard tools and a REST API for ease of operation and support. The relative simplicity was demonstrated when the entire system transitioned from PNNL to BNL in May of 2018, resulting in no issues to users and effectively zero downtime. The demonstrated performance is well above the needs for Monte Carlo simulations and the current limited data-taking, and is expected to handle all Belle II operations once full data-taking has commenced.

References

1. Squid project, “Squid” [software]. Available from http://squid-cache.org [accessed 2018-10-22]
2. Payara project, “Payara” [software]. Available from http://www.payara.fish [accessed 2018-10-22]
3. Swagger project, “Swagger” [software]. Available from http://swagger.io [accessed 2018-10-22]
4. PostgreSQL project, “PostgreSQL” [software]. Available from http://www.postgresql.org [accessed 2018-10-22]
5. IBM, “General Parallel File System” [software]. More information at https://www.ibm.com/support/knowledgecenter/en/SSFKCN/gpfs_welcome.html
6. Docker project, “Docker” [software]. Available at https://www.docker.com/ [accessed 2018-10-25].
7. NGINX project, “NGINX” [software]. Available at http://www.nginx.com [accessed 2018-10-22].
8. basf2 project, “basf2” [software], arXiv:1809.04299.
9. ROOT project, “ROOT” [software]. Available at http://root.cern.ch [accessed 2018-10-22].
10. Python project, “python” [software]. Available at http://www.python.org [accessed 2018-10-22].
11. Prometheus project, “Prometheus” [software]. Available at http://Prometheus.io [accessed 2018-10-22].
12. Grafana project, “Grafana” [software]. Available at http://grafana.com [accessed 2018-10-22].
13. Gatling project, “Gatling” [software]. Available at http://gatling.io [accessed 2018-10-22].
14. CVMFS project, “CERN Virtual Machine File System” [software]. Available at http://cernvm.cern.ch/filesystem [accessed 2018-10-22].