PROCEEDINGS OF SCIENCE

Development of a Scout Job Framework for Improving Efficiency of Analysis Jobs in Belle II Distributed Computing System

Hikari Hirata* **for the Belle II computing group**

Graduate School of Science, Nagoya University,
Furo-cho, Chikusa-ku, Nagoya, Aichi 464-8602, Japan

E-mail: hirata@hepl.phys.nagoya-u.ac.jp

At the Belle II experiment, massive collision data is collected to broadly advance our understanding of particle physics. To store and process the data, a worldwide distributed computing system is utilized. This system is also used for data analysis. It is important to suppress failed jobs, which is one of the causes that prevent the efficient physics analysis in this system. Because these failed jobs are originated from problems in the analysis script or improper settings of the job parameters, we developed a syntax checker and a scout job framework. The former detects syntax errors at the language level of analysis scripts and notifies the end-user before the job submission. The latter submits a small number of scout jobs with the same analysis script to process a small number of events prior to massive job submissions to detect complicated syntax errors or improper settings of job parameters. Only if the scout jobs succeed, the main jobs are submitted. As a result of the introduction of these two features, we can reduce system troubles and waste of computational resources due to failed jobs. This is also beneficial for end-users because the pre-test can be streamlined.

International Symposium on Grids & Clouds 2022 (ISGC 2022)
21 - 25 March, 2022
*Online, Academia Sinica Computing Centre (ASGC), Taipei, Taiwan****

*Speaker

PoS(ISGC2022)029

1. Introduction

The Belle II experiment [1, 2] is an advanced B-factory experiment using an electron-positron collider, SuperKEKB [3]. Our goal is to broadly advance our understanding of particle physics, e.g. search for physics beyond the Standard Model, precise measurements of electroweak interaction parameters, and exploration of properties of the strong interaction. Collision data began to be collected with the Belle II detector in 2019, aiming to acquire 50 times more data than the predecessor experiment, Belle [4]. At the end of the data taking, an order of 100 PB disk storage and tens of thousands of CPU cores are required [5]. To store and process the massive data with the resources, a worldwide distributed computing (DC) system is utilized. This system is also used for physics analysis. To perform physics analysis efficiently with this system, it is important that jobs are processed quickly without interfering with the analysis activities. A job which does not run properly due to issues hereafter a "failed job" is one of the causes of reduction in the efficiency. To suppress failed jobs, a syntax checker and a scout job framework were developed. In this paper, overviews of our system and physics analysis are given in section 2. The two features are described in sections 3-4, and their total performance is described in section 5.

2. Belle II DC System and Physics Analysis

The Belle II DC system consists of several pieces of software [5, 6]. The core one is DIRAC interware [7]. It provides a complete grid solution that interconnects end-users and heterogeneous computing resources. It includes a Workload Management System (WMS), a Data Management System, support of analysis job execution, and so on. Its extension BelleDIRAC has been developed to meet the requirements of the experiment, e.g. an automatic system that generates jobs for producing simulation samples and processing raw data [8]. For the distribution of data produced by the automatic system, Rucio, an advanced scientific Distributed Data Management system, is used [9, 10]. These software are distributed to computing resources and end-users by CVMFS [11]. In addition, FTS, AMGA, and VOMS are used as a file transfer service, a metadata service, and a virtual organization management service, respectively [12–14].

Processes for all aspects of the data-processing chain can be performed by the Belle II Analysis Software Framework (basf2) [15]. Their modular functions are controlled with Python 3 scripting. To analyze data distributed over computing sites, a client tool to support basf2 job execution on the DC system (gbasf2) and a set of client tools to manage submitted jobs and output data are provided. End-users can submit a group of basf2 jobs by a single command specifying input data sets and the other job parameters. In detail, the jobs are stored in JobDB when the command is executed as shown in figure 1. The parameters are quickly analyzed by several tasks called executors, e.g. site assignment based on input data, etc. What is done by all of the executors is henceforth referred to as job optimization. The jobs are registered in TaskQueueDB by the last executor. Gradually, WMS submits pilot jobs to the computing elements (CE) of the sites where input data is hosted. When the pilot job is submitted to the worker node and executed, the end-user job is pulled and input data is downloaded from the storage element (SE). After the basf2 process is finished, the output is uploaded to SE and the job is done. The end-user can download the output file for further analysis using the client tools. This workflow indicates that even if there is a problem with an

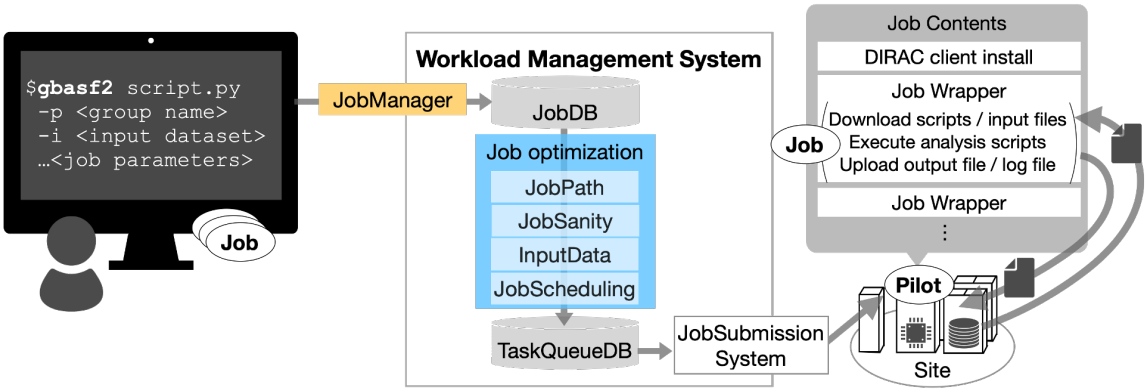


Figure 1: Workflow of usual analysis job submission on DC system. In the gbasf2 command, script.py shows the analysis script. Blue box shows a executor, and the job optimization is performed using four executors. Orange box and cylinder show a service and a database (DB), respectively.

end-user job, the job spends a few minutes on the worker node to authenticate and download input data. The failed jobs have potential to prevent efficient analysis in two points. First, worker nodes are unnecessarily occupied for at least a few minutes due to failed jobs. Second, when many jobs are submitted at once and they fail quickly, access to the central system, CE, and SE is concentrated for a short time. It often triggers system trouble and reduces system uptime. In addition, solving the trouble becomes a load on the operation side.

In 2019, 9.6 million analysis jobs were executed, but about ten percent of them failed. The main reason was not problems in the DC system or computing resources but basf2 or gbasf2 termination, which is caused by syntax errors, improper job parameter settings specified by the end-user, failure to upload output files, etc. At Belle II, therefore, python syntax checker and scout job framework were introduced as countermeasures against such failed jobs.

3. Python Syntax Checker

To prevent jobs with simple errors from being submitted, the python syntax checker is added to gbasf2. The syntax checker, e.g. open parenthesis and quotation, is realized by compiling the analysis script in the local environment with the existing python module, `py_compile`. This can detect syntax errors at the language level of analysis scripts before storing the job in the system. When a syntax error is found, it informs the end-user by displaying a message shown in figure 2. Only when the end-user agrees to submit, the jobs are stored in JobDB.

However, the problem is that basf2 and gbasf2 have different python environments, e.g. python version. If `py_compile` is executed for an analysis script under the gbasf2 environment, there is a possibility that incompatible syntax could be detected as an error, despite the correct syntax in the basf2 environment. Therefore, while executing gbasf2, another process with the same environment as basf2 is created, and `py_compile` is executed in that process.

```
$ gbasf2 testSteering.py -p testProject -i /belle/MC/release-05-02-00/DB00001
330/MC14ri_d/prod00021758/s00/e1003/4S/r00000/mixed/mdst/sub00/mdst_000237_pr
od00021758_task10020000240.root -s release-06-00-03
************************************************
  File "testSteering.py", line 12
      print("test)
                    ^
                    SyntaxError: EOL while scanning string literal

************************************************
Please fix the SyntaxError.

Are you sure to submit the project?
Please enter Y or N:
```

Figure 2: Example of output when the syntax checker detects an error

4. Scout Job Framework

Since the syntax checker is not enough to detect complicated syntax errors or the improper settings of job parameters, a scout job framework was developed. The targets of the framework are job groups with a large number of jobs. The criterion of the large number of jobs is configurable and currently sets as more than 100 jobs. Its concept is that original jobs and a small number of test jobs (henceforth referred to as "main jobs" and "scout jobs", respectively) are stored in JobDB at the same time, and the main jobs are submitted to sites only when scout jobs are successful. In addition, to avoid reducing the analysis activity, these tests were aimed to be completed more quickly. To achieve this functionality, a mechanism to generate scout jobs and a mechanism to control the submission of main jobs are necessary.

For the mechanism to generate scout jobs, the client tool was updated. To realize a more realistic test, scout jobs are made by copying some of the instances of the main jobs so that they have the same analysis script and the same input data. Moreover, to realize a quicker test, the number of events to be processed is reduced, and the number of the scout jobs is configurable and currently set to 10. In order to distinguish scout jobs from main jobs, different job types are set. Scout jobs are set to "UserScout", while main jobs as normal jobs are set to "User". In addition, some information about the scout jobs, e.g. an identifier to show the corresponding relation between scout jobs and main jobs, and a flag indicating whether scout jobs are completed or not, are registered in JobDB as attributes of the main jobs, that are utilized by the second mechanism described below. Here, the identifier consists of the first and the last jobID of the scout job group.

For the mechanism to control main job submission, two DIRAC components were newly added. The first component is a new executor described in section 2. Its role is to temporarily hold back the main jobs from registering in TaskQueueDB. At DIRAC WMS, it can be done by changing the job status to a new status that is not picked up by the DIRAC executors. This status is set to "Scouting". Considering these, its concrete behavior is to change the status to "Scouting" for jobs that have a corresponding scout job and the scouting has not been completed. The second component is a new agent, which is to perform actions periodically. Its role is to monitor the status of scout jobs and to take an action to resume the remaining job optimization of main jobs only when the scouting is judged to be successful. At DIRAC WMS, when setting jobs status as "Checking" for major state and an executor name for the minor status, the job optimization can be resumed from the executor.

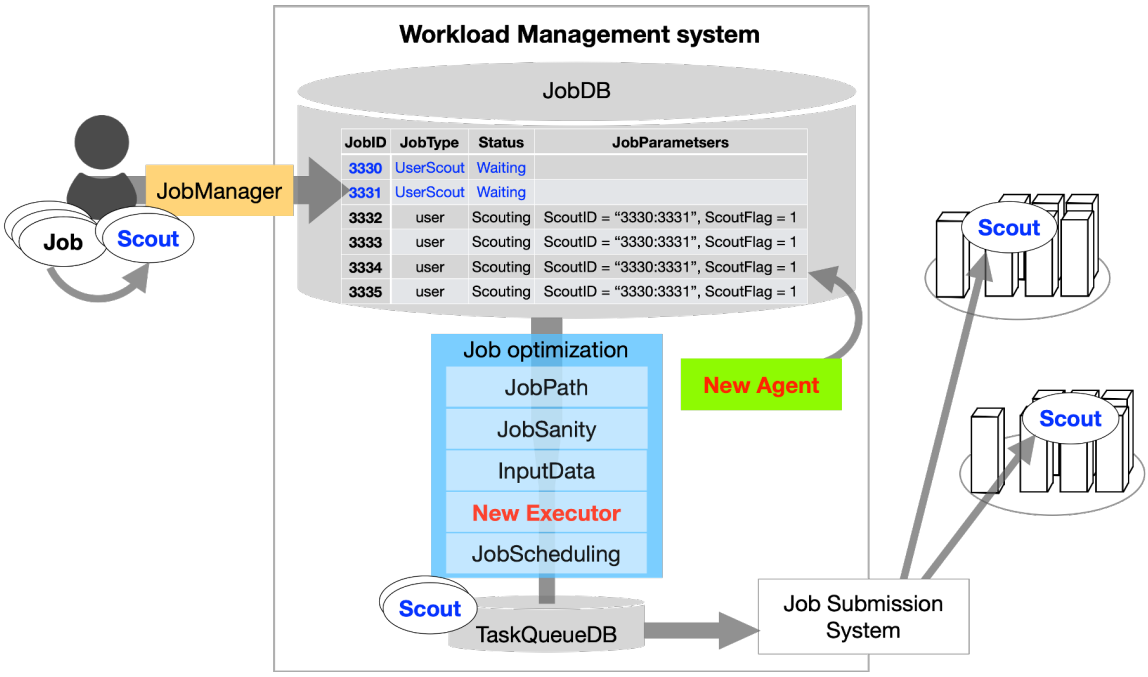


Figure 3: Design and workflow of the scout job framework

Concrete behavior is to search for jobs in "Scouting" status, monitor the final status of scout jobs related to them, and change the status of the main jobs according to the scouting results. The criterion for determining whether a group of scout jobs is successful is configurable. Currently, the criterion is set that at least 3 out of 10 scout jobs are successful. While this has the advantage of speeding up scouting, it has the disadvantage of not being effective in preventing job groups where some of the jobs fail due to site problems. This is discussed in more detail in the next section.

The overall workflow is shown below and in figure 3.

1. The client tool makes instances of main jobs and scout jobs. It registers both of them in JobDB
2. The new executor changes the status of the main jobs to "Scouting" to suspend job optimization, while the scout jobs go through all of the executors and the last executor registers them in TaskQueueDB
3. WMS submits pilot jobs and the scout jobs are pulled and executed on the worker nodes.
4. The new agent monitors the status of the scout jobs.
5. If the scouting fails, the agent changes the status of the main jobs to "Failed", and does nothing anymore. If the scouting is successful, the agent changes the status of the main jobs to "Checking" so that the job optimization is resumed and the main jobs are to be pulled for execution at the computing sites

From the end-user perspective, no action is required to use this framework. When an end-user wants to check the status of scout jobs, it can be done with the DIRAC Job Monitor or client tools,

☐	171693262	Done	Execution Complete	Prod_test_scout_1026	User
☐	171693261	Done	Execution Complete	Prod_test_scout_1026	User
☐	171693260	Done	Execution Complete	Prod_test_scout_1026	User
☐	171693259	Done	Execution Complete	Prod_test_scout_1026	User
☐	171693258	Done	Execution Complete	Prod_test_scout_1026	UserScout
☐	171693257	Done	Execution Complete	Prod_test_scout_1026	UserScout

(a) When scouting is successful

☐	171693547	Failed	Failed in scouting	Prod_test_scoutFail_1026	User
☐	171693546	Failed	Failed in scouting	Prod_test_scoutFail_1026	User
☐	171693545	Failed	Failed in scouting	Prod_test_scoutFail_1026	User
☐	171693544	Failed	Failed in scouting	Prod_test_scoutFail_1026	User
☐	171693543	Failed	Application Finished With Errors	Prod_test_scoutFail_1026	UserScout
☐	171693542	Failed	Application Finished With Errors	Prod_test_scoutFail_1026	UserScout

(b) When scouting fails

Figure 4: Examples of how scout and main jobs appear on the DIRAC Job Monitor for a job group where scout jobs were submitted; the panels show the case where scouting is successful and the case where scouting fails. In each panel, the top four rows show the main jobs and the bottom two rows show the scout jobs. The columns indicate, from left to right, job ID, status, minor status, job group name, and job type.

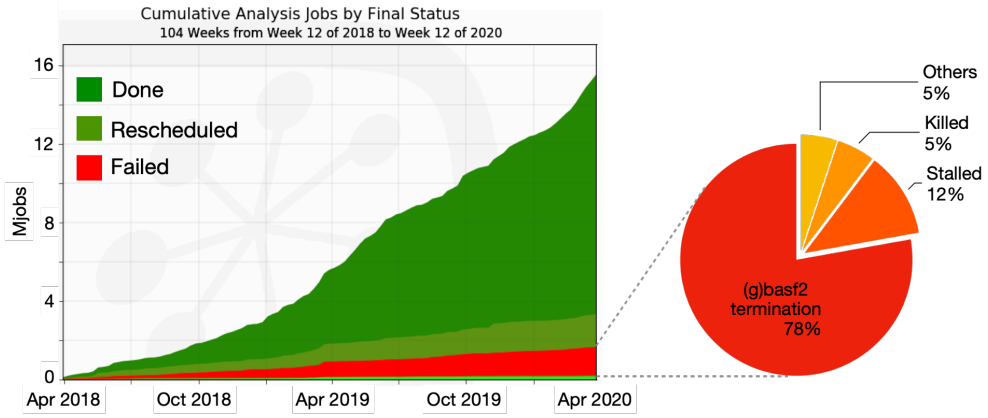
which are also used in the usual case. Figure 4 shows examples of how scout and main jobs appear on the DIRAC Job Monitor. End-users can distinguish scout jobs from main jobs by looking at the job types. In addition, end-users can download the log of the scout jobs from this monitor, so they can find out why they fail. Just in case the job has no problem but cannot be submitted due to a fatal bug of this framework, the client tool has an option to disable it from the end-user side.

5. Performance

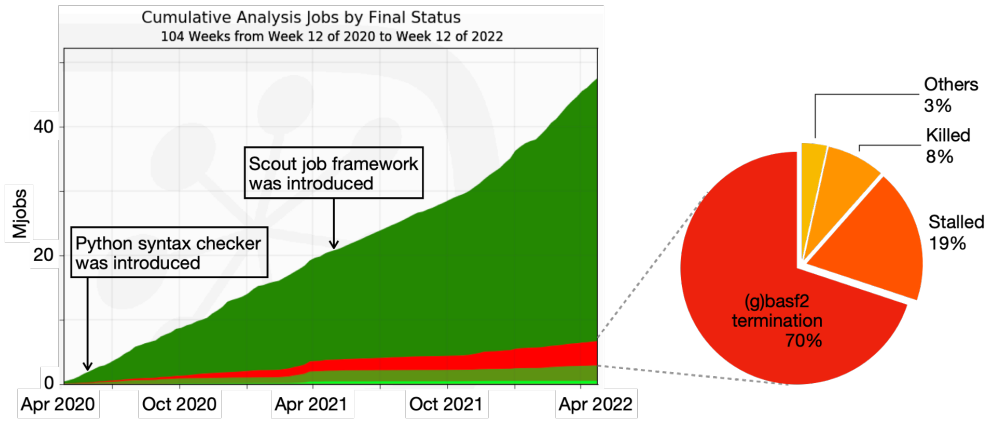
The python syntax checker and the scout job framework were introduced to the production environment in early May 2020 and late April 2021, respectively. Total performance was evaluated by comparing the percentage of failed jobs between two periods: two years before both introductions, Apr. 2018 - Mar. 2020, and those after the syntax checker was introduced first, Apr. 2020 - Mar. 2022. The left panels of figure 5 show the cumulative number of analysis jobs by job status in the two periods. Comparing these two plots, the total number of executed jobs in the latter period is three times larger than that of the former period. This is because analysis activity increased as data accumulated, and the number of active users increased by a relative 33%. The percentage of failed jobs is 9.5% and 7.9% in the former and the latter period, respectively. That results in a relative improvement of 17%. Under a more severe environment in which there are more people and opportunities to use the system, the situation could be improved. Pie charts of the job failure reasons in the two periods are shown in the right panels of figure 5. They show that the percentage of the basf2 or gbasf2 termination, which is caused by problematic analysis scripts and improper job parameter settings, was still high, although it decreased. This is expected because there are

many job groups whose scout job groups pass the success criterion when some of the scout jobs are successful and others fail.

Next, the performance of the scout job framework was evaluated. For this purpose, 26.7 million jobs executed in the period from April 2021 to March 2022 are analyzed. The number of job groups is 245 thousand. Of these, the 45.2 thousand job groups had more than 100 jobs, the target of this framework. 6.5% of them, 2930 groups, were deemed problematic. It means that at least 293 thousand failed jobs, which is the number of job groups multiplied by the target group criterion of 100 jobs, were suppressed. If such problematic jobs had been submitted, the number of failed jobs would have increased by more than 12% relative to the current results. The result concludes that this framework works well enough. However, the failure rate for scout jobs is 11.4%. In comparison, the percentage of the job group deemed problematic, i.e. the suppression rate, is low at 6.5%. It means that some job groups have passed the success criterion even though some of the scout jobs have failed. For example, it happens when there are problems in some of the computing sites.



(a) Results for two years before both introductions



(b) Results for two years after the syntax checker was introduced.

Figure 5: Cumulative number of analysis jobs by job status as a function of time and pie charts of the job failure reasons.

6. Conclusion and Prospects

To perform physics analysis efficiently with the DC system, it is important to suppress failed jobs. In Belle II, the python syntax checker and the scout job framework were developed and implemented to reduce failed jobs especially due to problematic analysis scripts or improper settings of the job parameters. As a result, the percentage of failed jobs was reduced compared to before these features were implemented. As for the scout job framework, it contributed to suppressing job groups that are clearly problematic, for example, when all scout jobs fail. Regarding the problematic job group, where some of the scout jobs fail, they were not adequately suppressed. It could be improved by tightening the success criterion of scouting. Thus, we could reduce the percentage of failed jobs, and the percentage of the basf2 or gbasf2 termination among them, even though the number of active users and user jobs has increased. This enables us to prevent failed jobs from causing system problems and waste of computing resources. In addition, users were asked to perform two types of tests before submitting jobs to the system: a local test of the analysis script and a test to submit a small number of jobs with the same analysis script to the system and check if they work. Thanks to these developments, the need for the latter test can be reduced, although local testing is still necessary. Thus, they also reduce the end-users effort.

Two further improvements are planned. The first one is to add a function to automatically correct problematic job parameters. This has potential to reduce stalled jobs caused by problematic job parameters. The second plan is to implement this framework into the automatic system that generates jobs for producing simulation samples and processing raw data. This would reduce the number of failed jobs in the entire system. These are going to bring us more efficient use of computing resources.

Acknowledgments

This work was supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research (S), No. 18H05226.

References

[1] E. Kou *et al.*, *The Belle II Physics Book*, *Progress of Theoretical and Experimental Physics*, **2019** (2019) 12, 123C01 [*Erratum ibid* **2020** (2020) 2, 029201] [arXiv:1808.10567].

[2] T. Abe *et al.* (Belle II Collaboration), *Belle II Technical Design Report*, KEK-REPORT-2010-1 [arXiv:1011.0352].

[3] K. Akai *et al.* (SuperKEKB Collaboration), *SuperKEKB collider*, *Nucl. Instrum. Meth. A* **907** (2018) 188-199 [arXiv:1809.01958].

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

[5] T. HARA on behalf of the Belle II computing group, *Computing at the Belle II experiment*, *J. Phys.: Conf. Ser.*, **664** (2015), 012002.

[6] Y. Kato on behalf of the Belle II computing group, *Overview of the Belle II Computing*, PoS(KMI2017)024.

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

[8] Hideki Miyake on behalf of the Belle II Computing Group, *Belle II production system*, *J. Phys.: Conf. Ser.* **664** (2015) 052028.

[9] M. Barisits *et al.*, *Rucio: Scientific data management*, *Comput Softw Big Sci* **3** (2019) 11 [arXiv:1902.09857].

[10] C. Serfon *et al.*, *Integration of Rucio in Belle II*, *EPJ Web of Conferences* **251** (2021) 02057.

[11] J. Blomer *et al.*, *The CernVM File System: v2.5.1 (2.5.1)*, https://doi.org/10.5281/zenodo.1472994.

[12] A A Ayllon *et al.*, *FTS3: New Data Movement Service For WLCG*, *J. Phys.: Conf. Ser.* **513** 032081.

[13] B. Koblitz *et al.*, *The AMGA Metadata Service*, *J Grid Computing* **6** (2008) 61–76.

[14] V. Ciaschini *et al.*, *Virtual Organization Membership Service (VOMS) (v2.0.14)*, https://doi.org/10.5281/zenodo.1875371.

[15] T. Kuhr *et al.*, *The Belle II Core Software*, *Comput Softw Big Sci* **3** (2019) 1 [arXiv:1809.04299].