Journal of Physics: Conference Series **664** (2015) 042041 doi:10.1088/1742-6596/664/4/042041

Improvement of AMGA Python Client Library for Belle II Experiment

Jae-Hyuck Kwak, Geunchul Park[1], Taesang Huh and Soonwook Hwang on behalf of the Belle II computing group[2]
Korea Institute of Science and Technology Information (KISTI)
245 Daehak-ro, Yuseong-gu, Daejeon 305-806, Republic of Korea

[2]https://belle2.cc.kek.jp/~twiki/pub/Public/ComputingPublic/AuthorList4Belle2Computing.tex

E-mail: jhkwak@kisti.re.kr

Abstract. This paper describes the recent improvement of the AMGA (ARDA Metadata Grid Application) python client library for the Belle II Experiment. We were drawn to the action items related to library improvement after in-depth discussions with the developer of the Belle II distributed computing system. The improvement includes client-side metadata federation support in python, DIRAC SSL library support as well as API refinement for synchronous operation. Some of the improvements have already been applied to the AMGA python client library as bundled with the Belle II distributed computing software. The recent mass Monte-Carlo (MC) production campaign shows that the AMGA python client library is reliably stable.

1. Introduction

The Belle II Experiment[1] is an upgrade of the B factory experiment Belle at the KEK laboratory in Tsukuba, Japan (where the charge-parity violation [CP-Violation] explaining why the universe today consists only of matter and no anti-matter, has been investigated). According to the timeline of the Belle II Experiment, it is expected that SuperKEKB accelerator commissioning will take place in early 2016, that the Phase 2 run (w/o the vertex detector) will start in 2017, and that the Phase 3 run (w/ the full detector) will commence in 2018. The Belle II Experiment is expected to produce about 100PB (one set of raw data) and approximately another 100PB (Monte-Carlo [MC]/analysis data) additionally. The computing scale of the experiment has attained 1.8GB/s at storage and tens of millions of files distributed across multiple grid sites. For this reason, the Belle II Experiment has been adopted as a distributed computing system based on Grid services[2].

AMGA (ARDA Metadata Grid Application), developed as a Grid metadata catalogue, has been adopted by the Belle II Experiment for distributed metadata management. We have been in contact with the Belle II computing group regarding the enhancement of AMGA functionalities for fulfilment of the metadata requirements of the Belle II Experiment. We were drawn especially to the action items

[1] Corresponding author

for improvement of the AMGA python client library after several in-depth discussions with the developer of the Belle II distributed computing system. It includes client-side metadata federation support in python, DIRAC SSL library support and API refinement for synchronous operation. Some of the improvements have already been applied to the AMGA python client library as bundled with the Belle II distributed computing software. The recent mass MC production campaign shows that AMGA python client library is reliably stable.

The rest of this paper is organized as follows. AMGA is introduced in Section 2. In Section 3, the Belle II distributed computing system is briefly described, and the role of AMGA as a metadata management system is explained. In Section 4, the improvement of the AMGA python client library is outlined in detail, focusing on why it was needed and how it was tackled. Finally, in Section 5, conclusions are drawn.

2. Overview of AMGA

AMGA (ARDA Metadata Grid Application)[3][4] is a stand-alone Grid metadata catalogue for support of metadata description, discovery and archiving of large-scale scientific data. It has been developed both to provide access to metadata of files stored on the Grid and to simplify DB access on the Grid. The key features of AMGA are as follows.

- Directory-like hierarchical structure
- Various authentication methods (ID/password, VOMS certificate)
- ACL-based authorization
- Heterogeneous DB back-ends (Modular back-ends: PostgreSQL, MySQL, Oracle, SQLite)
- Standardized access methods (Modular front-ends: TCP streaming, SOAP WS-DAIR, SSL)
- Multi-process/Multi-thread DB connection
- Pre-existing DB import
- Metadata replication and federation (experimental)
- General-purpose AMGA Manager GUI
- Various programming APIs (C++, Java, Python)

Figure 1 shows the AMGA metadata interface. Its basic concepts are collection, entry, attribute and schema. Each one is defined as follows.

- Collection: a set of entries associated with a schema
- Entry: the name of the data item or resource being described
- Attribute: a (key, value) pair with type information
- Schema: a group of attributes

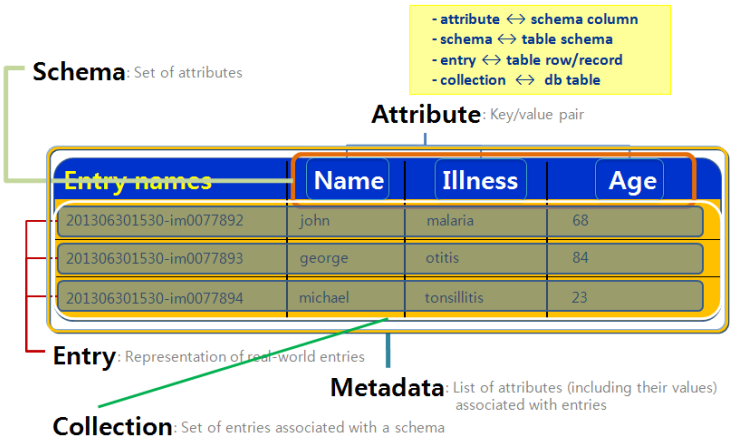

Figure 1. Metadata interface in AMGA

AMGA implementation uses a file-system model for structuring of metadata. Schemas play the role of directories: they contain entries and other schemas, allowing users to create a hierarchical structure. Entries are associated with a schema and inherit the attributes defined in it.

Figure 2 shows the main components of AMGA implementation. AMGA supports multiple database back-ends including PostgreSQL, MySQL, Oracle and SQLite. It offers two access protocols for clients: TCP streaming and WS-DAIR. The TCP streaming interface was implemented based on a streamed line-oriented ASCII protocol that provides for light-weight, high-performance communication between the client and the server. The WS-DAIR interface, the OGF standard for access to relational DBs, effectively addresses interoperability issues by allowing AMGA access to different databases by means of a standard protocol.

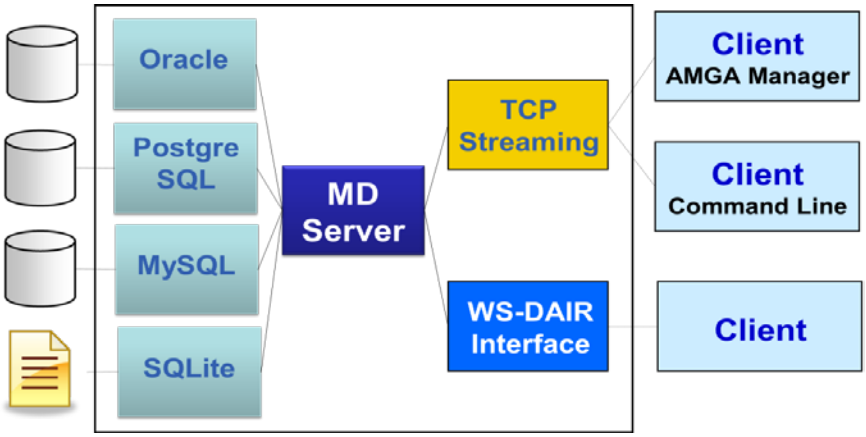

Figure 2. Main components of AMGA implementation

AMGA offers two types of advanced metadata functionalities: replication and federation. Though both of them are still at the experimental level, they might prove useful in providing required scalability, performance or fault-tolerance.

Metadata replication handles metadata redundancy, ensuring high metadata availability. For this purpose, AMGA uses an asynchronous, master-slave model in which write operations are allowed only at the master node. The master node maintains a log of all updates executed at its back-end. This log is shipped to the slave nodes that repeat the updates to keep themselves up-to-date. In order to avoid network latency, read operations might be off-loaded to the slave nodes that are closer to the applications.

Metadata federation serves metadata load balancing, thus ensuring the high performance of metadata. AMGA provides users with a virtualized view of all metadata, as if one metadata server held all of that data (which are actually distributed among multiple sites). This concept is very similar to that of an NFS directory that allows users access to data as if they were located at the local site. AMGA provides two types of metadata federation method: server-site federation and client-side federation. In server-side federation, the AMGA server does everything for the actual federation; thus no special API is needed on the client-side. In client-side federation, the client-side API does the actual federation; at present, however, it works only with AMGA C++ client API.

3. AMGA as metadata catalogue in Belle II distributed computing system

Figure 3 shows the computing model and workflow of the Belle II Experiment. Raw data from the detector is processed and stored by the main data centers, and mDST real data is produced. MC production is performed on the Grid sites as well as the Cloud resources, and produces mDST MC data. For the physics analysis, ntuples are generated using the mDST real and MC data. Finally for the ntuple analysis, the local sites are utilized.

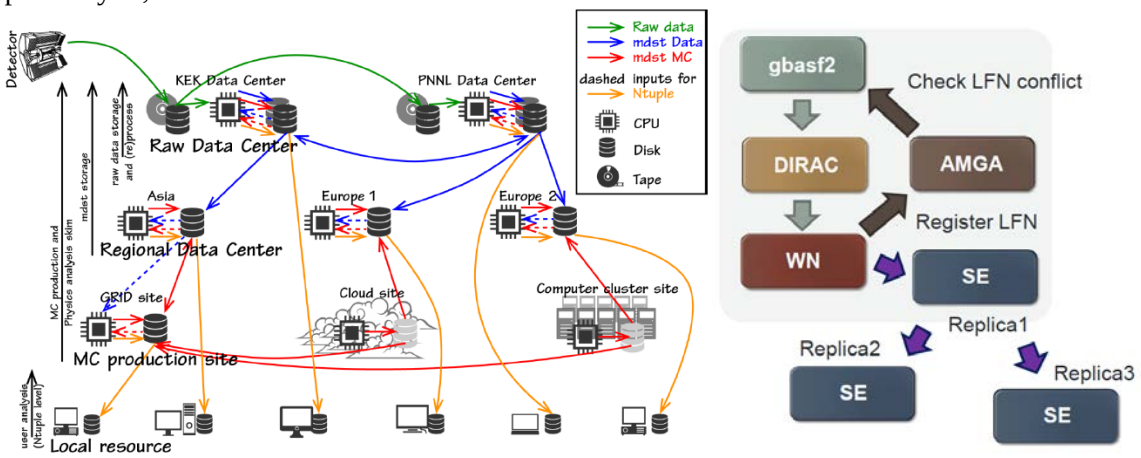

Figure 3. Computing model and workflow of Belle II Experiment

The Belle II distributed computing system consists of three main components: DIRAC[5] for the workload management system, AMGA[6] for the metadata catalogue, and gBasf2 for the job submission client.

DIRAC is a complete Grid solution for a community of users needing access to distributed computing resources. DIRAC is basically a pilot-based system; its concept allows for aggregation of computing resources of different kinds, including computational Grids, Clouds or clusters, transparently for users. AMGA is specialized in metadata management, as noted in Section 2, and supports advanced metadata functionalities such as metadata replication and metadata federation. gBasf2, a distributed analysis client for the Belle II Experiment, utilizes DIRAC and AMGA for workload and metadata management, respectively.

Before submitting a job to the Grid, gBasf2 asks AMGA to check for any LFN (Logical File Name) conflict, which is to say, to determine if the LFN already exists in the metadata catalogue. Subsequently, the gBasf2 job can be submitted by DIRAC to the Grid, and is executed on the WN at Belle II site. After the gBasf2 job has been completed, the output files are transferred to the SE, and finally, the LFN for each output file is registered to the AMGA along with the other metadata attributes.

4. Improvement of AMGA python client API

AMGA python client API lacks functionalities compared with AMGA C++ client API. The developers of the Belle II distributed computing system asked for an improved AMGA python client API, because python is the main programming language used by gBasf2. In this section, we describe up-to-date changes that were made to AMGA python client API. Those changes entailed client-side metadata federation support, DIRAC SSL library support, and API refinement for synchronous operation.

4.1. Support for Client-side Metadata Federation in Python

Client-side metadata federation in AMGA has several limitations. First, it is supported only by C++ client API. Second, it creates an entry for root ownership only, not also for non-root ownership. Finally, it supports only password-based authentication. These drawbacks limit the usability of client-side metadata federation.

We successfully implemented client-side metadata federation in AMGA python client API. The implementation details are located in the new python module (*mdfed.py*); additionally, we modified the main python module (*mdclient.py*) so that users can use the new module transparently. In the python implementation, ownership of the entry created by a non-root user is preserved by supporting certificate-based authentication.

Figure 4 illustrates an ownership preservation problem in client-side metadata federation. The left box indicates that the entry created by the belle2 user in the federated directory has root ownership, not belle2 ownership. The right box, by contrast, shows that the entry created by the belle2 user in the federated directory has the correct ownership.

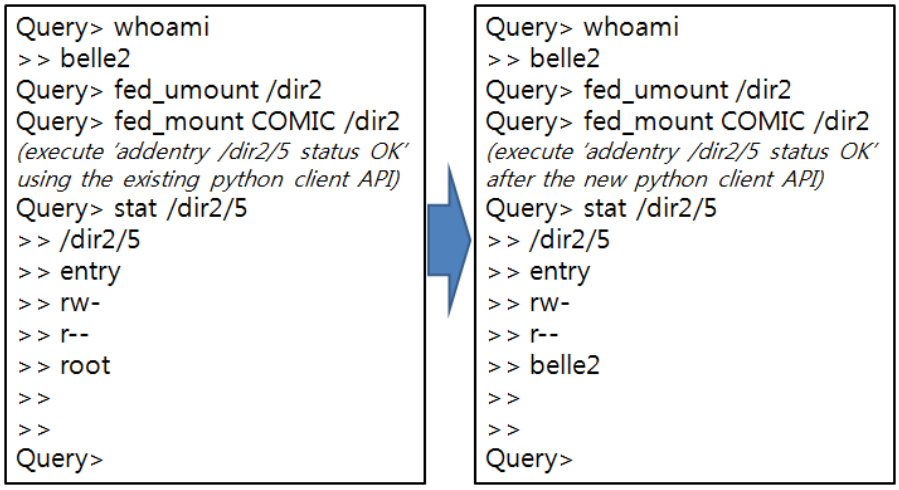

Figure 4. Ownership preservation in client-side metadata federation in python

4.2. Support for DIRAC SSL Library

gBasf2 uses different SSL libraries between DIRAC and AMGA. As shown in Figure 5, when it communicates with DIRAC, it uses GSI developed by DIRAC. If, on the other hand, it needs to communicate with AMGA, it uses the built-in SSL library in python or TLS Lite, a third-party python SSL/TLS library. This utilization of different SSL libraries by gBasf2 complicates the use of the Belle II distributed computing system by developers or users.

We added GSI support to AMGA python client API. Now, AMGA python client API supports three SSL library implementations: built-in SSL, TLS Lite, and GSI. To enable GSI, *USE_GSI* flag in the main module (*mdclient.py*) needs to be turned on. We observed that with the newly added GSI support in AMGA python client API, the stability of the SSL connection during the MC campaign was improved.

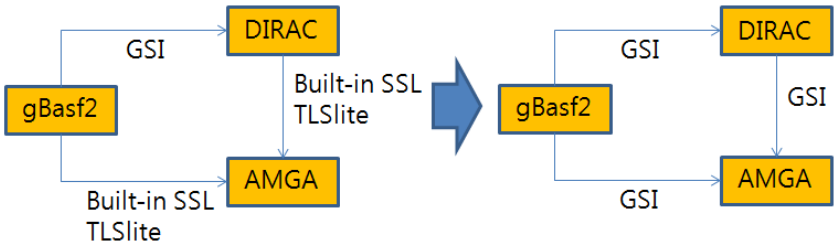

Figure 5. SSL library usage in gBasf2 workflow

4.3. Support for Synchronous Operation by API refinement

AMGA python client API has two stages, execute and fetch, of communication with the AMGA server. The execute stage sends a user command to the AMGA server, and the fetch stage obtains the result from the AMGA server. This asynchronous operation makes AMGA python client API more prone to misuse by gBasf2.

In order to hide the two separate stages to users and support synchronous operation, we refined AMGA python client API. Specifically, we modified three APIs in the main module (*mdclient.py*), namely *listEntries*, *selectAttr*, and *getAttr*, using the locking mechanism with the *with* statement in python.

Figure 6 provides examples of the usage of the new APIs with synchronous operation as compared with the existing APIs. As can be seen, each API has a control flag, *SYNC_FLAG*, as the last argument. If synchronous operation is required, the flag's value should be true. These changes make it easier and more convenient to use AMGA python client API.

listEntries:

```
client.listEntries('/gilda/movies')
while not client.eot():
  file, type=client.getEntry()
  print "->", file, type[0]
```

```
entries=client.listEntries('/gilda/movies', true)
print entries
['/gilda/movies/1', '/gilda/movies/2']
```

getattr:

```
client.getattr('/gilda/movies', ['title', 'LFN'])
while not client.eot():
  file, values=client.getEntry()
  print "->", file, values
```

```
attries=client.getattr('/gilda/movies', ['title', 'LFN'], true)
print attries
{'1':['Spiderman', 'file://spiderman'], '2':['Batman', 'file://batman']}
```

selectAttr:

```
client.selectAttr(['/gilda/movies:title', '/gilda/movies:LFN'], '')
while not client.eot():
  values=client.getSelectAttrEntry()
  print "selected ->", values
```

```
values=client.selectAttr(['/gilda/movies:title', '/gilda/movies:LFN'], '', true)
print values
[['Spiderman', 'file://spiderman'], ['Batman', 'file://batman']]
```

Figure 6. Examples of usage of new APIs with synchronous operation

We added another new API, *selectQuery*, to support conditional querying in AMGA python client API. Similar APIs, *directQuery* and *directQueryWithAttributes*, exist in an AMGA high-level API that was developed as a practical interface for gBasf2. These we generalized into the single API, *selectQuery*. Figure 7 shows an example of how the *selectQuery* API is used. Whereas the process is similar to that for the *selectAttr* API, references to attributes to be queried are passed by a *list* object.

```
selectQuery:
values=client.selectQuery('/gilda/movies', ['FILE', 'title', 'LFN'], 'FILE>₩'1₩')
print values
[['2', 'Batman', 'file://batman']]
```

Figure 7. Example of usage of selectQuery API

5. Conclusions and future work

AMGA is a Grid metadata catalogue used to deal with large-scale metadata generated from the Belle II Experiment. We derived the action items for library improvement after in-depth discussions with the developer of the Belle II distributed computing system. We improved AMGA python client API and provided, here in this paper, the implementation details. We hope and expect that these changes will be helpful and beneficial to future efforts to enhance the stability and usability of AMGA for the purposes of the Belle II Experiment.

Acknowledgments

We are grateful for the support and the provision of computing resources by CoEPP in Australia, HEPHY in Austria, McGill HPC in Canada, CESNET in the Czech Republic, DESY, GridKa, LRZ/RZG in Germany, INFN-CNAF, INFN-LFN, INFN-LNL, INFN Pisa, INFN Torino, ReCaS (Univ. & INFN) Napoli in Italy, KEK-CRC, KMI in Japan, KISTI GSDC in Korea, Cyfronet, CC1 in Poland, NUSC, SSCC in Russia, SiGNET in Slovenia, ULAKBIM in Turkey, UA-ISMA in Ukraine, and OSG, PNNL in USA. We acknowledge the service provided by CANARIE, Dante, ESnet, GARR, GEANT, and NII. We thank the DIRAC and AMGA teams for their assistance and CERN for the operation of a CVMFS server for Belle II.

References

[1] "Computing at the Belle II experiment", T. Hara, CHEP2015 proceedings.
[2] "Belle II production system", H. Miyake, CHEP2015 proceedings.
[3] "AMGA", http://amga.web.cern.ch/amga
[4] N. Santos and B. Koblitz "Metadata services on the grid", *Proc. Advanced Computing and Analysis Techniques*, 2005
[5] A. Casajus, R. Graciani, S. Pateson, A. Tsaregorodtsev and the LHCb DIRAC team "DIRAC pilot framework and the dirac workload management system", *J. Phys. Conf. Ser.* 219, 062049, 2010.
[6] S. 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.