AstroDS - A Distributed Storage for Astrophysics of Cosmic Rays. Current Status[*]

Alexander Kryukov[1][0000−0002−1624−6131], Igor Bychkov[2], Elena Korosteleva[1], Andrey Mikhailov[2][0000−0003−4057−4511], and Minh-Duc Nguyen[1][0000−0002−5003−3623]

[1] M.V.Lomonosov Moscow State University, D.V.Skobeltsyn Institute of Nuclear Physics, Russia
kryukov@theory.sinp.msu.ru
[2] Matrosov Institute for System Dynamics and Control Theory, Siberian Branch of Russian Academy of Sciences, Russia

Abstract. Currently, the processing of scientific data in astroparticle physics is based on various distributed technologies, the most common of which are Grid and cloud computing. The most frequently discussed approaches are focused on large and even very large scientific experiments, such as Cherenkov Telescope Array. We, by contrast, offer a solution designed for small to medium experiments such as TAIGA. In such experiments, as a rule, historically developed specific data processing methods and specialized software are used. We have specifically designed a distributed (cloud) data storage for astroparticle physics data collaboration in medium-sized experiments. In this article, we discuss the current state of our work using the example of the TAIGA and CASCADE experiments. A feature of our approach is that we provide our users with scientific data in the form to which they are accustomed to in everyday work on local resources.

Keywords: Astroparticle physics · Distributed data storage · Metadata.

1 Introduction

The modern physics of astroparticles is one of the most rapidly developing areas of modern science. It includes several scientific fields, in each of which a number of large experimental installations have been put into operation. So, in the field of gamma astronomy, the HESS [1] and MAGIC [2] experiments are working, and in 2023 the CTA installation [3] is to be commissioned. In the field of neutrino astrophysics, we note IceCube [5], as well as the ongoing Global Neutrino Network (GNN) [6], The Baikal deep underwater neutrino telescope (or Baikal-GVD - Gigaton Volume Detector) [8], which will include IceCube,

KM3NeT [7] and Baikal-GVD [8]. In the field of physics of high energies of cosmic rays, we note The Pierre Auger Observatory [9]. The LIGO-Virgo consortium [10, 11] sets a new direction in the study of gravitational waves.

However, in addition to the experimental megascience installations mentioned above, there are also medium and small installations. An example of such installations is the TAIGA [12, 13], TUNKA [14] installations deployed in the Tunkinskaya valley in Buryatia (Russia), the KASCADE [15] installation and many others.

These installations also collect a large amount of data during their operation. An important feature of these experiments is that analysis of their data is based on long-established practices using specific data processing methods and customized software. On the other hand, there is an insistent need to properly preprocess the collected data and make them accessible through a web service with a convenient and user-friendly interface. Putting it all together, it is important to provide web access to the data while maintaining the ability to work with the data using existing software and techniques.

This study was carried out within the framework of the Russian-German initiative [19] aimed at supporting the processing of data from astrophysical experiments throughout the entire data life cycle: from collection and store to the preparation of data analysis results for publication and data archiving.

Of course, several approaches to the design of distributed data storages have already been proposed earlier. One of the most striking examples is the global system GRID [20], which was originally created to store and process LHC data, and later came to be used for many other experiments.

The International Virtual Observatory Alliance (IVOA) [21] sets similar tasks. Another example is the CosmoHub [22] system, which is based on the Hadoop distributed storage [23].

All of these experiments use large project-oriented approaches that, for various reasons, may not be suitable for medium and small experiments.

Another important trend in astronomy is the combined analysis of data from various sources (multi-messenger astrophysics) [24], which is used to obtain a more detailed physical picture of the observed high energy astrophysical phenomena. In particular, a comparison of how the same phenomenon was observed by different small experiments could yield interesting new results. Making such a comparison requires the development of shared cloud storage for small experiments.

Thus, the development of cloud storage for small experts is an urgent task.

This article describes an approach to creating such a cloud storage and providing convenient access to data in it. The created cloud storage is called AstroDS. The storage is focused on medium to small sized experiments such as TAIGA. The work is a logical continuation of the work of A.Kryukov with co-authors [25].

The structure of the article is as follows. In Section 2, we provide a brief description of the principles that were used as the basis for the development of a data storage. The third section is devoted to some peculiarities of working

with remote storages based on data storage using relational databases using the example of KCDC [16]. The fourth section describes the AstroDS cloud storage prototype and its main characteristics. In conclusion, we discuss the results obtained and the plan for the further development of the cloud storage.

2 Design and architecture of the AstroDS

As we said, AstroDS cloud storage is focused on small and medium experimental collaborations. This left a certain imprint on the decisions that were made during the system design process.

The main principle underlying the development of AstroDS cloud storage was the principle of maximizing preservation of the historically established methods of user interaction with local storages. Thus, the main requirements were:

– preservation of the structure of data directories;
– an opportunity to mount directories on local computers;
– data transfer over the network should occurs at the time of real flicking to data.

This approach makes it possible to practically eliminate the modification of application software when working with cloud storage. At the same time, the load on the network is minimized.

Another very important goal was to make it as easy as possible to integrate existing local storages as a cloud storage node. At the same time, the load on the local storage equipment should not significantly increase to ensure collaboration in the cloud storage. To achieve this goal, an approach was chosen when all user requests are processed off-line on a special server that stores all the metadata necessary for data retrieval. The collection of metadata is performed at the time of data loading to local storage.

Note that the storage implements a two-level data selection architecture:

– search at the file level, for example, by session number;
– search at the level of individual events in files, for example, by the energy of the event.

This solution is flexible enough to fulfill almost any user request.

A special case is the integration of those local storage that store data on a per-life basis in a relational database. An example of such a storage is the KCDC [16] Data storage of the KASCADE [15] Experiment. This case will be considered in more detail below.

In AstroDS all user requests are processed asynchronously, which can be important in some cases when you need to prepare a large sample.

Both the web interface and the command line interface are available to users. The latter mode is more convenient when using a set of scripts to automate the data processing.

Taking into account all the above requirements, we have developed an architectural solution shown in Fig. 1.
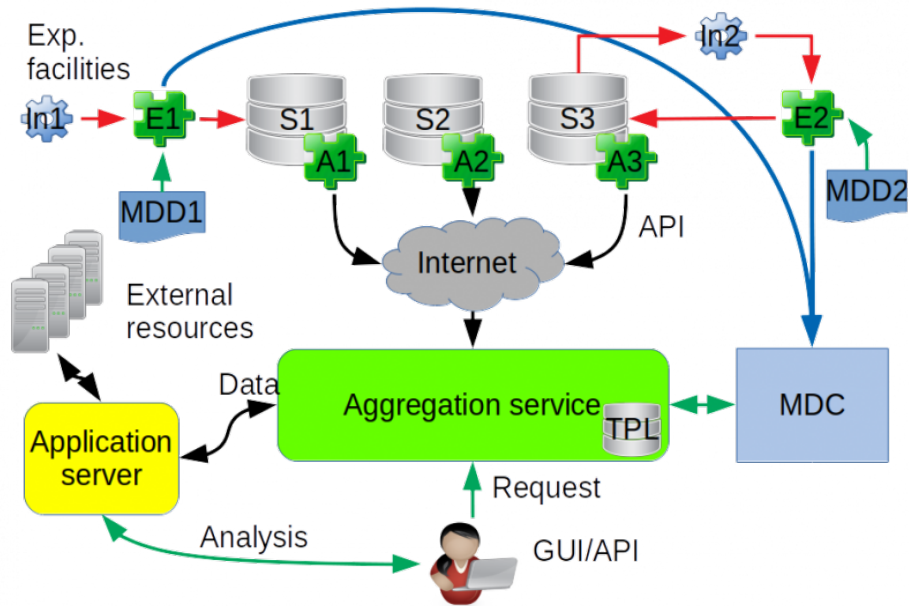
Fig. 1. Simplified architecture of AstroDS.

Below we will focus on a number of individual features of the implementation of the developed architecture. A more detailed presentation of the general principles of building the AstroDS system can be found in the articles [19, 25].

3 Metadata catalog

The Metadata Catalog (MDC) is a single place where the physical location of the requested data is determined. The MDC is a service which supports two main functions:

– register collected metadata;
– process the user requests for data.

3.1 Metadata catalog API

The MDC architecture is based on the integration of several standard solutions (see Fig. 2). To store metadata we chose Timescale DB [26] – a special database for storing time-series data. Flask [27] is used as a web server for user requests processing. To enable the aggregation service to interact with the MDC, an API

was implemented using GraphQL [28] query language. We used the Graphene-Python library [29] to easily create GraphQL APIs in Python. SQL Alchemy as object relation mapper for TimeScale DB.
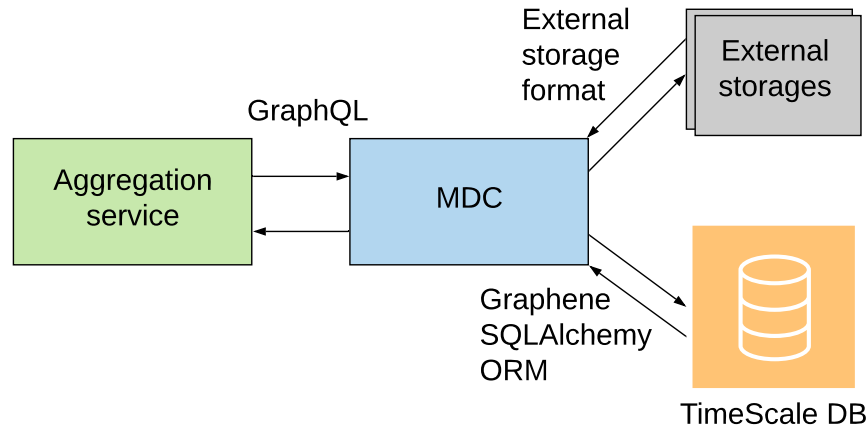

Fig. 2. The architecture design of the MDC

MDC provides an API for data insertion and for searching using the filter list shown in Table 1. We do not provide an API for updating data in storage and deleting data from storage because the main idea of APPDS is that metadata is extracted only once by a special program called an extractor. All insert operations are implemented by a special GraphQL type - mutation.

Parameter name	Description
fid	Unique file identifier
startTime	Event start time
endTime	Event end time
first	Count of data for pagination
offset	Start position for pagination
weatherId	Weather at the time of observation
trackingSourcId	Observed object
facilityId	The facility that captured the event

Table 1. Avaliable MDC parameters

The query structure shown in Listing 1 consists of two main parts - data fields and query parameters. The data fields correspond to the DB schema and include such information as the run date, cluster, weather, facility, etc. All these data fields are primarily intended for the aggregation service, the end-user is interested in the url to download the file. The list of available query parameters allows you to filter data by event start and end time, facility, weather, tracking source, and unique file identifier.

Listing 1. The query structure

```
query{
   files([query parameters]){
      [data fields]
   }
}
```

A GraphQL integrated development environment was deployed to test the API through the web GUI. An example of the GraphQL response is shown in Fig. 3.

```
{
  "data": {
    "files": [
      {
        "fid": "98771",
        "fullPath": "/k2/DATA_IACT/2017-18/dec17/201217/BSM08/20127008.029",
        "runDate": "2017-12-20 00:00:00",
        "facilityId": 1
      },
```

Fig. 3. Example of GraphQL response

3.2 Filters for data selection

The MDC can store data from different facilities. Each facility has its own parameters for filtering data. In order for the aggregation service to form a list of filters for the client, the formal specifications for each facility are stored in the metadata catalog. Specifications include a list of options available for a given facility. Parameters can have one of five data types: "date", "int", "float", "string", "list". The first four parameters are the base data types. Type "list" could be a query string or an array of base types. The query string is required when the filtering value is contained in the database table. In this case, the aggregation service makes a request by this query string to the metadata catalog to get a list of parameters.

Specifications are stored in JSON format for each facility. In Listing 2 an example of the specification is shown. The start time and end time have "date"

tipe and comparison sign as equal. The third parameter "weather" has type "list" and contains querying string.

Listing 2. Filters specification

```
{
  "filters": "[
     {\"name\": \"startTime\", \"type\": \"datetime\",
      \"conditions\": [\"=\"]},
     {\"name\": \"endTime\",   \"type\": \"datetime\",
      \"conditions\": [\"=\"]},
     {\"name\": \"weather\",   \"type\": \"list\",
      \"options\": { \"table\": \"weather\",
         \"request\": \"query{weather{id wScale}}\",
         \"fields\": {\"id\": {\"type\": \"integer\"},
            \"wScale\": {\"type\": \"string\"}},
         \"conditions\": [\"=\"]
      }
    }
  ]"
}
```

4 Integration with KCDC

Sometimes access to raw data is not available. In this case, there is no way to extract the metadata and save it to the metadata catalog. For such cases, MDC is used as a proxy for user requests. In this case the third-party storages get request directly and process it themselves.

If third-party storages does not have a compatible query format with MDC API the MDC converts it in proper format and wise versa. So the aggregation service works with a uniform query format independently of storage API. For this purpose the only new converter should be added to MDC. This module describes how to translate formats to each other.

MDC knows all available external storages and their API formats. When MDC gets a request from aggregation service it translates GraphQL request to external request format and sends it and waits for the response. After, it translates the response back to GraphQL and sends it to aggregation service.

One example of how it works is shown in the Listing 3 and Listing 4. For this request, where facility id means request KASCAD data, fields start time and end time are converted from date-time format to timestamp. The request is formed in JSON with additional fields. And after receiving the response, MDC prepares a unified response for the aggregation service with the URL to download the file. The availability of the file for download is checked on the side of the aggregation service.

Listing 3. GraphQL request

```
query{
```

```
    files(
        facilityId: 5,
        startTime: "2017−12−20T00:00:00"
        endTime: "2017−12−21T00:00:00"
    ){
        url
        facilityId
        __typename
    }
}
```

Listing 4. JSON–RPC 2.0 request

```
{
    "id": "cf919bb3−7028−4238−9438−2601cbfde3a6",
    "jsonrpc": "2.0",
    "method": "new_task",
    "parameters": {
        "type": ['kascade'],
        "datetime_min": 1256481198,
        "datetime_max": 1256481201
    }
}
```

5   The Aggregation service

The aggregation service (see Fig. 1) is the central service of the AstroDS system
and a single point of user entry into the system. Its main tasks are as follows:

- building of user requests and their transfer to MDC;
- requesting data from the local storages based on the MDC advice and pro-
  viding the data requested by the user in the form of files and / or mount
  point of the virtual file system;
- selection of events that meet the criteria of user requests from the received
  files and the formation of new subsets based on them.

The main requirement when developing an aggregation service is to mini-
mize network traffic and load on remote data stores. For this, files stored on
remote storages are mounted on the aggregation service as a virtual file system
CVMFS [18]. This file system initiates real data transfer only at the moment of
actual data access.

Generating subsets of events that meet the criteria from a user request is a
tedious task. Using the aggregation service for this purpose minimized interfer-
ence with the work of local storages and relieve them of the unpredictable load
on their resources.

The aggregation service is described in more detail in the work of Nguyen et
al. [17].

6 AstroDS testbed

The AstroDS system is currently operating in a test mode at the test site, which includes three storage facilities. Two of them are real data storages of the TAIGA and KASCADE experiments, and one is the testers storage. Thus, the general structure of the polygon is as shown in Figure 4.
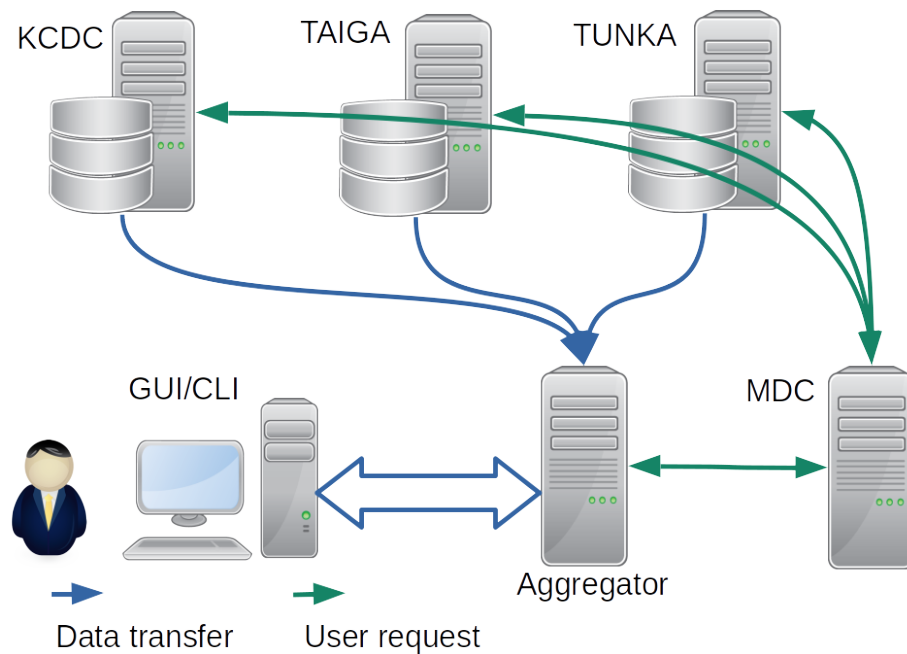

Fig. 4. AstroDS test bed

The user can send requests through the aggregation service (Aggregator). The type of requests is still limited to file-level requests, but with the release of the update to version 2, the system will work both with file-level requests and requests that require a selection of individual events according to user criteria. The main problem associated with event-level queries is the presence of specific meta-information, which, as a rule, arises after primary data processing or further analysis.

Note that among the three local storages integrated into the system, only the KCDC storage puts data event-by-event in a relational database. This is reflected in the way we handle user requests. Namely, since it is not known in advance whether there are events that meet the requirements in the user's request, then

all requests, regardless of their content, are sent to KCDC and processed there. As a response, KCDC always provides a link to a file (zip) that contains the selected events. This file will be provided to the user either for download or for mounting on his work computer in the form of a virtual file system. If there are no required events, KCDC will send a link to an empty file.

7   Conclusion

The article presents the results of the development of a distributed cloud data storage, called AstroDS, for medium and small astrophysical experiments. It was shown that the principles underlying the construction of such a storage make it possible to practically exclude modification of application software and preserve the usual methods of working with data.

The deployed prototype of the AstroDS system currently includes data from the TAIGA, TUNKA and KASCADE experiments.

Integration of data from several experiments allows them to be used for joint analysis (multi-messenger), which will increase the accuracy of such analysis and the possibility of studying new phenomena.

In the future, it is planned to expand the functionality of the AstroDS cloud storage both in the direction of increasing the flexibility of data selection and in the number of experiments integrated into the system.

Acknolegement

References

1. HESS, https://www.mpi-hd.mpg.de/hfm/HESS/. Last accessed Aug. 30, 2020
2. Rico, J. for the MAGIC Collaboration: Overview of MAGIC results. Nuclear and Particle Physics Proceedings **273–275**, 328–333 (2016)
3. Cherenkov Telescope Array. Exploring the Universe at the Highest Energies. https://www.cta-observatory.org/. Last accessed Aug. 30, 2020
4. The IceCube Neutrino Observatory. https://icecube.wisc.edu/. Last accessed Aug. 30, 2020
5. The IceCube Neutrino Observatory. https://icecube.wisc.edu/. Last accessed Aug. 30, 2020
6. The Global Neutrino Network (GNN). https://www.globalneutrinonetwork.org/. Last accessed Aug. 30, 2020
7. KM3NeT the next generation neutrino telescopes. https://www.km3net.org/. Last accessed Aug. 30, 2020
8. The Baikal deep underwater neutrino telescope (or Baikal-GVD – Gigaton Volume Detector). https://baikalgvd.jinr.ru/. Last accessed Aug. 30, 2020

9. The Pierre Auger Observatory. https://www.auger.org/. Last accessed Aug. 30, 2020

10. . https://www.ligo.org/. Last accessed Aug. 30, 2020

11. . https://www.virgo-gw.eu/. Last accessed Aug. 30, 2020

12. TAIGA. https://taiga-experiment.info/. Last accessed Aug. 30, 2020.

13. Budnev, N. and etc. The TAIGA experiment: From cosmic-ray to gamma-ray astronomy in the Tunka valley. Nuclear Instruments and Methods in Physics Research. Section A, **845**(2017), pp.330–333, https://doi.org/10.1016/j.nima.2016.06.041

14. Hiller R., Budnev N.M. et al. Status and First Results of Tunka-rex, an Experiment for the Radio Detection of Air Showers. Phys.Procedia, **61**, 708-713 (2015)

15. W.D.Apel and etc. The KASCADE-Grande experiment. Nuclear Instruments and Methods in Physics Research, Section A, **620**(2010), pp.202–216, https://doi.org/10.1016/j.nima.2010.03.147

16. KASCADE Cosmic Ray Data Centre (KCDC). https://kcdc.ikp.kit.edu/. Last accessed Aug. 30, 2020.

17. Nguyen M.-D., Kryukov A., Mikhailov A. The current design and implementation of the AstroDS Data Aggregation Service. In: The IV International Workshop "Data Life Cycle in Physics" (DLC-2020). This proceedings.

18. Welcome to CernVM-FS's documentation! https://cvmfs.readthedocs.io/en/stable/. Last accessed in Aug. 20, 2020

19. Bychkov, I., et al.: Russian–German Astroparticle Data Life Cycle Initiative. Data, **4**(4), 56 (2018). DOI: 10.3390/data3040056.

20. Worldwide LCH Computing GRID. http://wlcg.web.cern.ch/. Last accessed Aug. 30, 2020.

21. The International Virtual Observatory Alliance (IVOA). http://www.ivoa.net/. Last accessed Aug. 30, 2020.

22. Tallada, P., et al. CosmoHub: Interactive exploration and distribution of astronomical data on Hadoop. Astronomy and Computing. **32**, 100391 (2020).

23. Shvachko, K., Kuang, H., Radia, S., Chansler, R., The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). 1–10 (2010). doi:10.1109/MSST.2010.5496972.

24. Mészáros, P., Fox, D.B., Hanna, C. et al. Multi-messenger astrophysics. Nat Rev Phys **1**, 585–599 (2019). https://doi.org/10.1038/s42254-019-0101-z

25. A.Kryukov et al., Distributed Data Storage for Modern Astroparticle Physics Experiments. In: The III Int. conf. DLC-2019, CEUR-WS, **2406**, 78–83 (2019). http://ceur-ws.org/Vol-2406/paper9.pdf, ArXiv:1907.06863

26. Timescale: Time–series data simplified. https://www.timescale.com/. Last accessed Jul. 16, 2020

27. Flask. https://palletsprojects.com/p/flask/. Last accessed Jul. 16, 2020

28. GraphQL – A query language for your API. https://graphql.org/. Last accessed at Jul. 16, 2020

29. Graphene – Python. https://graphene-python.org/. Last accessed at Jul. 16, 2020