PROCEEDINGS OF SCIENCE

GAMAS - a Generic And Multipurpose Archive System

E. Lyard[a], R. Walter[a], V. Sliusar[a], D. Neise[b], A. Biland[b] and P. Fernandez[c]

[a] *Université de Genève, Département d'Astronomie, Chemin d'Ecogia 16, 1290 Versoix, Switzerland*

[b] *ETH Zurich, Institute for Particle Physics and Astrophysics, Otto-Stern-Weg 5, 8093 Zurich, Switzerland*

[c] *Swiss National Supercomputing Centre, Via Trevano 131, 6900 Lugano, Switzerland*

E-mail: etienne.lyard@unige.ch

There exist many distributed storage systems that are mature and reliable. These systems are not fully adapted to the needs of an open astroparticle community mainly because they do not follow the Open Archival Information Systems (OAIS) standard. Moreover they require adaptations by each of the data centres at which they are installed. We introduce GAMAS, a novel distributed OAIS that tackles the problem of different technologies used at the various DCs. Instead of imposing the requirements of GAMAS to the DCs, we allows them to simply provide a python interface (or plugin) to their storage. This allows GAMAS to be easily deployed on top of different architecture and technologies, and to transparently allow users to retrieve data for processing, wherever they are. A metadata browsing system is incorporated within GAMAS and allows DCs as well as anonymous users to retrieve datasets based on high-level queries. GAMAS' central database can be reconstructed from the archived data, which makes the system robust against corruption. We will expose the current status, architecture and functionalities of GAMAS and also detail the current test case which stores about 0.6 PB of data from the FACT experiment at separate data centres.

36th International Cosmic Ray Conference -ICRC2019-
July 24th - August 1st, 2019
Madison, WI, U.S.A.

PoS(ICRC2019)731

1. Introduction

There exist many distributed storage systems that are mature and reliable. Among this jungle of approaches and implementations, DIRAC [1], iRods [2] and OneData [3] can be named as of particular interest for the science community. Despite a large offer, none of these systems are fully adapted to the needs of an open astroparticle community. The main reason being that they are not compliant with the Open Archival Information Systems (OAIS) standard. Another reason is that they require data centres (DCs) that run them to accommodate for their specific needs. Even if such systems are commonly used in experiments, retrieving the data requires expert knowledge and privileged rights. For instance accessing data on DIRAC requires to be the holder of a GRID certificate [4]. Consequently, experiments data are rarely distributed to the community and often remain accessible only by a small group of users.

To address these recurrent shortcomings, we introduce GAMAS, a novel distributed Open Archival Information System. GAMAS is a lightweight python package that can be interfaced with any storage via a plugin system. GAMAS is modular and can be used stand-alone or in conjunction with a pre-existing distributed storage system. Our main motivation to develop GAMAS was to have:

- A system that ensured data integrity and provide high-level data management based on experiments metadata independently of specific Data Center infrastructure or technology.
- Is very light-weight (7500 lines of code) such that it could be maintained by its users for a low cost.
- Allow anonymous users to browse and retrieve archived data.

GAMAS does not operate on files but rather on datasets which are a collection of files that have a logical relationship between them.

GAMAS is organised around a Central Database Management System (C-DBMS) which stores the basic metadata related to archived datasets. Further metadata is made available to users in a separate Metadata DBMS (M-DBMS). Both the C-DBMS and the M-DBMS can be reconstructed from the archived data, thus ensuring that the system is robust. The experiment's metadata browsing system is based on W3Browse [5] which is maintained by NASA/HEASARC and hosts data from most high energy space missions. Other interfaces could be interfaced as well especially if they are database driven. It allows DCs as well as anonymous users to retrieve datasets based on high-level queries.

This paper first goes through the overall architecture of GAMAS (section 2). It then digs more in details in its functionalities in section 3. In section 4 we will explain how storage systems can be interfaced and in particular what are our plans to interface GAMAS with DIRAC (section 4.1). Eventually we report on the status of the current prototype in section 7 and future plans in section 8.

2. Architecture

GAMAS is composed of a set of instances orchestrated by a Shepherd, via the C-DBMS, as seen on figure 1. It can be interfaced to W3Browse for experiments metadata queries and retrieval.

W3Browse has its own M-DBMS which is separated from the C-DBMS because of performances, flexibility and reliability considerations. Different experiments data can either use the same C-DBMS or each have its own. If they should use the same storage backend, then one instance per experiment should be created, with shared storage and separate databases. It is also possible that there is more than one instance at a given site, for a single experiment to increase the total throughput.

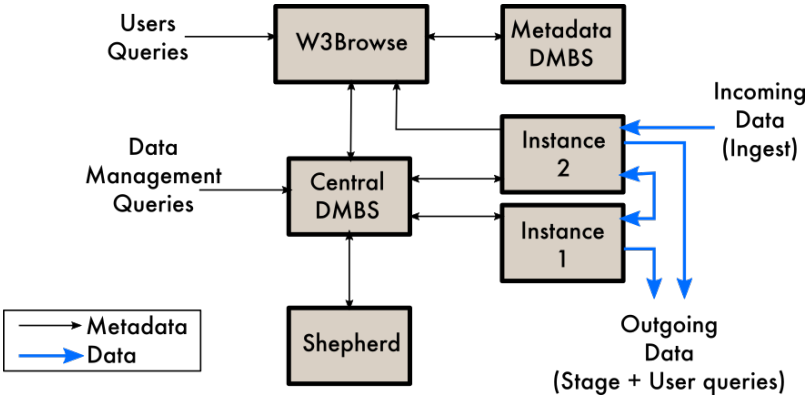

Figure 1: Overall architecture of GAMAS with 2 instances.

Each instance is interfaced to a specific set of storages as seen on figure 2. The storage interface is actually the same for all storages, only the implementation differs depending on the backend. Actual data only flow to or from the instances which reduces the load on the Shepherd and C-DBMS. Each instance provides the same set of functionalities namely *Ingest*, *Expose*, *Copy* and *Stage*. At least one instance must run an ingest process, which verifies, archives and registers incoming datasets. Ingested datasets are then replicated until the requested number of copies exists in the system. Each instance must be provided with some compute cores and some local storage on the same node as the compute cores. This can be achieved in a number of ways, either by providing physical hardware (possibly with quotas), or through a virtual machine or container. Any input or output operation go to or from the local storage so that interfaces to storage systems remain simple.

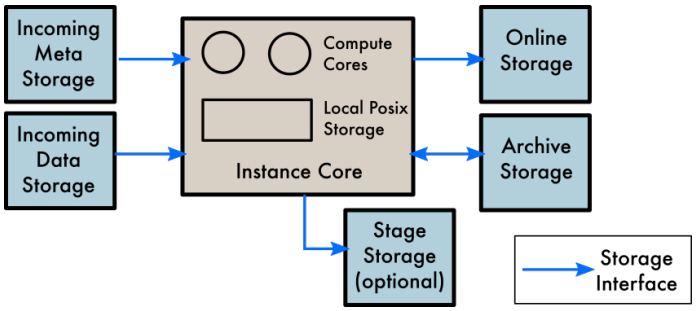

Figure 2: Internal architecture of a GAMAS instance. The core of the instance is composed of some compute cores and local storage. This core is interfaced to various storage systems via storage interfaces. Note that it is not required that a storage be at the same physical location as the core of the instance.

Experiments metadata is extracted during the ingest process and registered in the M-DBMS. Upon a distribution query by a user, W3Browse registers the requested datasets in the C-DBMS

Keyword	Purpose	Example
experiment	Apply specific experiment related rules.	FACT
data_type	Apply specific data-type related rules.	RAW
date	UTC Timestamp to be attached to the dataset.	2019-01-22T23:00:00.000
short_id	String to be concatenated to the dataset ID. Only the dataset ID needs to be unique.	20190122_130.fits.fz
script	Optional script to extract experiment-specific metadata. Can contain arguments.	FACT_MDIF_extraction.py
local_path	Where to get the input data in the incoming data storage.	raw/2019/01/22/ 20190122_130.fits.fz
storage_type	Preferred storage type for archived copy(ies).	NEARLINE, ONLINE or OFFLINE
req_num_copies	Desired number of duplicates. Achieved only if storage quotas allow for it.	1, 2, 3...
checksum	SHA256 checksum of the dataset. If the dataset consists of more than one file, this checksum is the sum of the sha256 of each file in the dataset.	sha256:ee4b66dbecde18f8ab5...

Table 1: Required fields in the input metadata.

and they are then taken care of by GAMAS alone. Anonymous queries will remain online for a fixed period of time, while housekeeping queries are removed as soon as they are completed. Housekeeping queries are made directly to the C-DBMS, either by the Shepherd or by an operator.

3. Functionalities

Each GAMAS instance provides three core functionalities: *Ingest*, *Expose* and *Copy* plus an optional one called *Stage*. Below is a description of each of them:

Ingest is the process that adds new datasets to the archive. A dataset is a collection of files, which ingestion is triggered by the creation of a specific file at a predefined location. Upon detecting the trigger file, the ingest process retrieves the dataset using metadata provided in a yaml file, along with the trigger file. This metadata is summarised on table 1. After retrieval, further experiment related metadata might be extracted depending on the configuration. This experiment-metadata extraction requires that a script provided by the experiment outputs an MDIF file, which will then be used to populate the W3Browse database.

The next step tars the dataset into a single tarball for easier handling. The storage implementation is free to keep the tarball as it is, for instance for tape storage, or to store the data untared, for instance on an online storage.

All metadata are included in the tarball. Once a new tarball is created, its checksum is verified against the one provided along with the dataset. The steps performed by the ingest pipeline can be run either sequentially or in parallel, while the number of datasets processed at once can be tuned by the user so that the usage of local resources remains under control.

Expose retrieves a given dataset from the archive storage. It then moves this dataset to the online storage, which is accessible via obfuscated https URLs. Obfuscated URLs were chosen because

they allow to perform access control: the obfuscation key is required to figure out the URL of a given dataset. The obfuscation key can be seen as a password to retrieve the dataset, which voids the need for users accounts. *Expose* is also used to perform the duplication a dataset (*Copy*) and to gather datasets at a specific location (*Stage*). Upon retrieving the dataset, its checksum is verified against its expected value. Upon failure of this operation, the dataset copy is removed from the database. This triggers the re-duplication of the dataset from another location. If no more copies are available an email is sent to the archive's maintainer. The actual archived data is not removed at that stage, in case the database itself is corrupt.

Copy involves two instances plus the Shepherd. Upon detecting that not enough copies of a given dataset are available, the Shepherd will trigger the copy mechanism. Source and destination are chosen based on user-defined rules, including the type of available storage and allowed locations for a given dataset. The source instance exposes the dataset which is then downloaded by the target instance. After downloading the dataset the target instance verifies its checksum again before storing it in its archive. If the checksum is wrong the download is attempted again until a max. number of trials is reached. The source copy is assumed to be valid, as the *Expose* process verified it.

Stage is very similar to Copy, except that the data is stored to a specific storage and untared. The goal of this functionality is to allow groups of datasets to be made available at a specific location for processing. If the staging happens locally then the download step is skipped. Stage is an operator-triggered action while copy is triggered by the Shepherd. It may be possible that some instances do not have this functionality available.

4. Interfaces to storage

IO operations to all storages are abstracted away via a simple API. The API seen by GAMAS is always the same while the implementation on the storage side can change depending on the storage type, backen and DC. New storage backends can be interfaced to GAMAS by providing a python class that implements the following methods: `store()`, `retrieve()`, `remove()` and `list()` and `get_modification_time()`. The details of their implementation is left to the storage administrator, which gives much flexibility in interfacing GAMAS to an existing infrastructure. Several interfaces are available already and can all be used simultaneously by an instance of GAMAS:

- **Shell** uses python shutil package to move data on filesystems mounted locally.
- **rsync** uses linux system calls to move data using rsync. This interface can access local and distant filesystems.
- **Openstack Swift** uses a python wrapper around the REST API of Swift [6].
- **IBM TSM** uses a TSM client [7] locally installed and configured on the machine running the instance.
- **OneData** uses the native OneData REST API [8].
- **S3** uses the Amazon S3 API [9].

Even though they are reusable when similar system are found elsewhere, they must be configured to authorise the GAMAS instance to actually access the resources. This is done in specific ways depending on the implementation. For instance the rsync interface requires that the user running GAMAS has a valid ssh key to access the remote host, while *Openstack Swift* needs a valid user and password stored somewhere on the local filesystem.

4.1 Interfacing to DIRAC

At the time of writing this text, we had just engaged in interfacing GAMAS to DIRAC. DIRAC is widely used at CERN and elsewhere, and already manages hundreds of Petabytes of data. As it is unnecessary to reinvent the wheel, our goal is to use DIRAC for all actual storage, replication and processing jobs. GAMAS would be somewhat reduced to only a gateway to and from DIRAC. GAMAS will still handle the Ingestion of new datasets, as well as the browse and retrieval of public and private datasets.

This will allow experiments to exploit the existing features of DIRAC along with the ease of configuration and ingestion of GAMAS. Moreover, users wouldn't need to get a GRID certificate to retrieve some datasets, as they will be able to do that via GAMAS instead. The full functionalities of DIRAC would remain available to expert users.

DIRAC has a file catalog that maps the datasets with their corresponding file while GAMAS doesn't. As we will gain more operational experience, and if it turns out that the DIRAC way of handling datasets is better, then we might modify the C-DBMS schemas accordingly.

5. Interfacing with W3Browse

W3Browse is the public front-end of GAMAS. It allows anonymous users to browse and retrieve datasets based on high-level, experiment related metadata. Browsing is done with W3Browse only, while the retrieval of datasets had to be interfaced with GAMAS. The users first perform queries on W3Browse. Once they are satisfied with the result of the query he/she enters his/her email address and hits the *retrieve datasets* button. This registers the requested datasets in the C-DBMS. The Shepherd then inspects the requests and orchestrates the staging of the requested datasets. Online datasets can then be downloaded by the user, directly from the instance's online storage. If not enough online storage is available to make all of the requested data available at once, the request is split-up into smaller chunks by the Shepherd. As soon as the first chunk is available then the Shepherd sends a notification email to the user. The email includes a download script that downloads all the datasets and untar them locally on his/her laptop. Datasets available online are kept for 1 week or until the user enters a specific completion code provided along with the notification email.

6. Configuration

Instances are configured via YAML files. Each configuration file features the URI of the C-DBMS, the name of the current instance, storage quotas per storage type, along with the configuration of all storage interfaces and processes. The storage configuration can use custom keywords, specific to a given type of storage. As an example here is a possible configuration for OpenStack

Swift:

```
incoming_data_interface:
      class_name:     gamas.io.SWIFT_interface.SwiftStorage
      init_args:
            auth_url:        "https://my.server.com:1234"
            id_provider:     "nsa"
            id_url:          "https://example.com/identity"
            project_id:      "1ab2c3d4e56f7g8h9ij"
            container:       "FACT_INCOMING_DATA"
            path_to_key:     "/some/local/path"
```

Only non-critical information should be stored here, as there is a high likelihood that these files are poorly secured, e.g. in a github repository. In the example above, only the path to the file storing the access key is given, not the key itself.

The processes configuration is more simple and consists of a list of local directories to be used by each process. Only the ingest process has extra parameters, namely whether it should be sequential or parallel, and how many datasets should be ingested in parallel.

7. Current prototype

The current prototype handles approximately 400TB of data from the FACT experiment [10] and 200TB more is being added. A schematic view of the current prototype can be seen on figure figure 3. The data is first transferred from the Observatorio del Roque de los Muchachos (ORM) on the island of La Palma. The first temporary storage in Europe is located at the University of Geneva (UNIGE). There it undergoes verification and next-day analysis. It is then pushed to one of the two instances of GAMAS. Once on the instance's local storage, the data is verified again before being moved to the long-term, cold storage and registered in the C-DBMS. We use a python script to create the metadata needed to ingest the data, while dedicated scripts are used to push the data to either instance. The exact same code runs on both instances, though with different configurations due to the different technologies available at each site.

The instance at UNIGE runs on bare metal, while the one at CSCS runs in a virtual machine with only 2 cores, 8GB of RAM and 1TB of local storage. Even though the resources allocated at CSCS are small, the ingest process can run at up to 200MB/s as long as enough throughput is available from the various storage interfaces. If the storage systems are fast enough, then the bottleneck is the recalculation of the checksum of the datasets being ingested.

Datasets can be staged back to a BeeGFS [11] filesystem within the internal network of the University of Geneva. This takes a single command to achieve. The journey of datasets being staged starts from the archive storage of their source DC. From there the data is moved to the local storage before being moved again to the online storage. From the online storage the GAMAS instance at UNIGE downloads it locally before rsynching it to its final location. The extra copy to the local filesystems might look unnecessary at first glance. This is not true because very often compute cluster are hidden on local networks, and cannot be accessed remotely. This extra hop

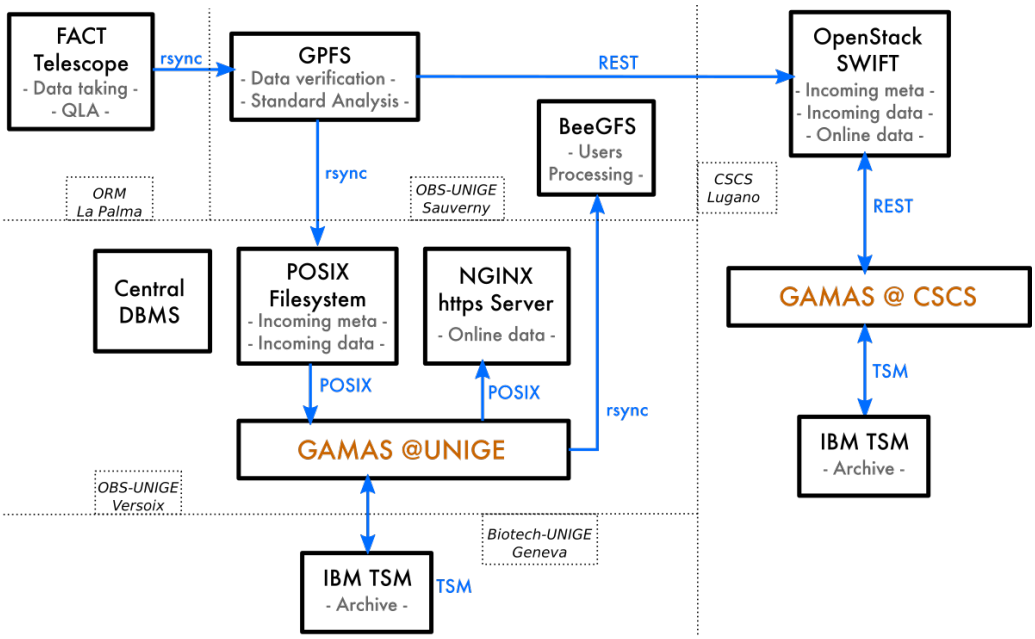

Figure 3: Overview of the current GAMAS prototype deployment.

between the instance storing the data and its final destination ensures that the data can be moved regardless of the security configuration of the compute cluster.

8. Future Work

We will continue to work on the current deployment before rolling out a new one for the SST1M telescope [12]. This new deployment will be shared between UNIGE and CYFRONET, one of the polish partners of the SST1M consortium. After that we plan to focus on interfacing GAMAS with DIRAC.

References

[1] DIRAC | The INTERWARE. http://diracgrid.org/ accessed June 2019

[2] IRODS. https://irods.org/ accessed June 2019

[3] ONEDATA - Global data access solution for science. https://irods.org/ accessed June 2019

[4] The Worldwide LHC Computing Grid. http://wlcg.web.cern.ch accessed June 2019

[5] W3Browse - NASA's archive system. https://heasarc.gsfc.nasa.gov/w3browse/ accessed June 2019

[6] Openstack Swift. https://wiki.openstack.org/wiki/Swift accessed June 2019

[7] IBM Tivoli Storage Manager. https://en.wikipedia.org/wiki/IBM_Tivoli_Storage_Manager accessed June 2019

[8] https://onedata.org/docs/doc/using_onedata/using_onedata_from_cli.html accessed June 2019

[9] https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html accesed June 2019

[10] H. Anderhub et al. Design and operation of FACT - the first G-APD Cherenkov telescope. Journal of Instrumentation, vol. 8, 2013.

[11] BeeGFS - The Parallel Cluster File System. https://www.beegfs.io/content/ accessed June 2019

[12] Schioppa E. et al - The SST-1M camera for the Cherenkov Telescope Array. in Proceedings of ICRC2015. PoS(ICRC2015)930