Key developments of the Ganga task-management framework

To cite this article: I A Dzhunov et al 2012 J. Phys.: Conf. Ser. 396 032061

View the article online for updates and enhancements.

Related content
- Reinforcing user data analysis with Ganga in the LHC era: scalability, monitoring and user-support
  Johannes Elmsheuser, Frederic Brochu, Ivan Dzhunov et al.
- Distributed Data Analysis in the ATLAS Experiment: Challenges and Solutions
  Johannes Elmsheuser and Daniel van der Ster
- Ganga: User-friendly Grid job submission and management tool for LHC and beyond
  D C Vanderster, F Brochu, G Cowan et al.
Key developments of the Ganga task-management framework

I A Dzhunov\textsuperscript{1}, M J Kenyon\textsuperscript{2}, J Moscicki\textsuperscript{2}, A J Richards\textsuperscript{3}, M W Slater\textsuperscript{4}, F Brochu\textsuperscript{5}, J Ebke\textsuperscript{6}, U Egede\textsuperscript{3}, H Lee\textsuperscript{7}, D C Van Der Ster\textsuperscript{2}, M J Williams\textsuperscript{3}, J Elmsheuser\textsuperscript{6} and M Jha\textsuperscript{8}

\textsuperscript{1}University of Sofia, 5 James Bourchier Blvd., 1664 Sofia, Bulgaria
\textsuperscript{2}European Organisation for Nuclear Research, CH-1211 Geneva 23, Switzerland
\textsuperscript{3}Imperial College London, London SW7 2AZ, UK
\textsuperscript{4}University of Birmingham, Edgbaston, Birmingham B15 2TT, UK
\textsuperscript{5}University of Cambridge, Trinity Lane, Cambridge CB2 1TN, UK
\textsuperscript{6}Ludwig-Maximilians-Universität, Geschwister-Scholl-Platz 1, 80539 Munich, Germany
\textsuperscript{7}Nikhef, Science Park Amsterdam 105, 1098 XG Watergraafsmeer, Netherlands
\textsuperscript{8}INFN, via Enrico Fermi, 40 - 00044 Frascati, Rome, Italy

E-mail: project-ganga-developers@cern.ch

Abstract. Ganga is the main end-user distributed analysis tool for the ATLAS and LHCb experiments and provides the foundation layer for the HammerCloud system, used by the LHC experiments for validation and stress testing of their numerous distributed computing facilities. Here we illustrate recent developments and demonstrate how tools that were initially developed for a specific user community have been migrated into the Ganga core, and so can be exploited by a wider user-base. Similarly, examples will be given where Ganga components have been adapted for use by communities in their custom analysis packages.

1. Introduction

Ganga is a user interface for creating analysis jobs and managing their execution across a range of computing resources (\textit{backends} in Ganga terminology), whether that be a single job running on a stand-alone computer or a batch of tasks exploiting the distributed computing power of the Worldwide LHC Computing Grid (WLCG)[1].

Irrespective of the backend used Ganga provides a flexible, Python [2] based programmatic interface which gives users the choice of interacting with the framework either interactively via the IPython shell, with custom Python scripts or from the Unix command line interface.

Users typically work with a particular Ganga application module that has been designed to function with the appropriate experiment software stack and computing resource infrastructure. ATLAS users, for example, use the Athena framework, an application skeleton into which developers can plug in their code. Within Ganga, the GangaATLAS package provides both the ability to use this framework, plus the necessary mechanisms to allow ATLAS users to access their data (whether local or remotely hosted) and submit analysis tasks to distributed computing resources. In addition, generic applications are provided which can be used to run arbitrary operating system commands (subject to the normal user-privilege restrictions) or user-supplied scripts and/or compiled code. Thus Ganga provides a level of abstraction for the end
user, insulating them from the complexity of their particular experiment framework, and thereby allowing them to focus on the real task of analysing their data. Furthermore, Ganga makes it simple to split a job into multiple subjobs such that each subjob runs with a different set of input parameters, or over a range of input data files.

The modular construction of Ganga provides an extensible framework onto which further components can be added, such as tools to submit jobs to additional backends, or applications designed for new user-community workflows. Ganga is a key tool of the LHC user communities, being both a popular user-analysis platform for ATLAS and LHCb, plus providing the basis of the HammerCloud test framework, used by the LHC experiments to validate their distributed computing resources.

This paper illustrates Ganga developments including improvements to the core Ganga code which provide for more powerful handling of applications and their associated files. We will also describe a new web-based monitoring interface (the Ganga WebGUI) that allows users to conveniently view the status of their submitted Ganga jobs and browse the local job repository. Finally, we detail the Ganga error-reporting tool, which was developed to help communities provide a quicker response time to their users’ support requests, and has since been integrated into the main user distributed analysis tool for the CMS experiment.

2. Ganga Core: A prepared state for applications
Ganga uses the concept of applications to define what should be executed. The simplest of these is the Executable application (see example, below), which allows users to run arbitrary code or system commands and optionally pass arguments and/or environment variables.

```
Out[5]: Executable (  
exe = 'echo' ,  
env = {} ,  
args = ['Hello World'] ,  
is_prepared = None )
```

Experiment specific application objects are, necessarily, more complex and can contain attributes representing configuration files and experiment framework options, as a sample of the custom ATLAS application (Athena) demonstrates:

```
Out[4]: Athena (  
atlas_project = '' ,  
glue_packages = [] ,  
useRootCoreNoBuild = False ,  
useNoDebugLogs = False ,  
run_event_file = '' ,  
max_events = -999 ,  
atlas_exetype = 'ATHENA' ,  
atlas_run_dir = './' ,  
stats = {} ,  
user_area_path = '' ,  
atlas_supp_stream = [] ,  
run_event = [] ,  
collect_stats_stream = False ,  
atlas_cmtconfig = '' ,  
exclude_package = [] ,  
append_to_user_area = [] ,
```
A typical workflow scenario using the illustrated Athena application involves at least three steps:

(i) Definition of the application attributes.
(ii) Preparing the application object.
(iii) Submission of the job.

The process of preparing an Athena application is initiated by the user and, behind the scenes, involved configuring the application parameters correctly and creating an archive of the required analysis framework files in a temporary location (i.e. the standard, Unix /tmp directory). Storing the file in a temporary directory has the obvious drawback that its existence cannot be guaranteed indefinitely. Furthermore, those using Ganga on the CERN Linux public login user service (which accounts for almost 80% of users) cannot even guarantee that they will have access to the same client PC on a long-term basis.

To work around these issues, a solution was developed which involved migrating the prepare functionality into the Ganga Core, from where it can also be exploited by a wider range of users, rather than only the ATLAS community. This new implementation was designed to be generic enough that it can be exploited from any Ganga application package, and to achieve this required the introduction of several new Ganga components, which are briefly described here.

**ShareDir: The Shared Directory object**

In order to provide a persistent means of storing an application’s analysis framework and/or
executable archives, the ShareDir directory object was developed. This is a mechanism for generating a randomly-named directory in a dedicated subdirectory of the user’s Ganga job repository which is, by default, located in their home directory.

New application attributes
During implementation of the prepared state, two new attributes were added to the Ganga internal application schema, and one meta-attribute. In turn, these were:

(i) The `is_prepared` attribute which signifies whether the application has been prepared; if it has, this attribute will contain a reference to the relevant ShareDir object, otherwise it will hold the `None` value.

(ii) The `hash` attribute, which is hidden from users and holds an MD5 digest of the application’s text representation, is used by Ganga to determine whether an application has been modified since it was originally prepared.

(iii) The boolean `preparable` meta-attribute which is attached to those application attributes that can potentially hold file objects to be copied into the ShareDir directory during preparation. In addition, after the preparation has completed, attributes with this meta-attribute set to `True` cannot be modified. Being a meta-attribute, it is hidden from users and is only used by internal Ganga methods.

ShareRef: The Shared Directory reference counter
Multiple applications may utilise the same ShareDir object and, for optimal use of storage, ShareDir directories which are no longer referenced by a prepared application (i.e. are orphaned) are deleted when Ganga exits. To keep track of the number of applications using a particular ShareDir object, a reference counting mechanism was implemented. Thus, when an application is prepared, an entry is created in the ShareRef table for the corresponding ShareDir and its counter initialised to 1. Then, each time that particular ShareDir is reused by an application, the counter is incremented (Figure 1).

Figure 1. The ShareRef reference counting table. Column 1 shows the names of ShareDir objects that are or have been used by prepared applications; column 2 is the date of ShareDir creation; column 3 shows how many prepared applications reference a particular ShareDir. Note that ShareDirs with a reference count of 0 (row 3 in the table) are removed at Ganga exit.

The application `prepare()` method
Each application that exploits the new functionality exports the `prepare()` method to the Ganga interface. Though specific to each particular application, the `prepare()` methods are responsible for copying file objects marked in the application schema as preparable into the appropriate ShareDir, and for incrementing the ShareRef counter as necessary. The
The final step of the `prepare()` method involves calculating an MD5 digest of the application’s preparable attributes, and storing the result in the application schema. This value is recalculated and compared against the initial value each time the application is written to the user’s Ganga repository. This allows warnings to be generated in the event that an application’s preparable (i.e. locked) attributes become changed post-preparation.

The developments described above provide the ability to easily save an analysis task and rerun it at a future date, with the added assurance that configuration parameters will not have been adjusted in the interim. Furthermore, multiple analysis jobs (perhaps running over different input datasets) can use the same prepared application and, since the application and its associated configuration files can be shared between many jobs, utilisation of the user’s storage is optimised.

3. WebGUI
Ganga provides several methods for monitoring the progress of submitted jobs. To complement the long-standing IPython interface, which presents a flexible text-based output format (Figure 2), an HTML-based interface has been developed which can be enabled by the user at Ganga startup. Hosted by a simple HTTP server on the Ganga client computer, the user navigates their browser to the indicated local URL in order to view their Ganga job repository (Figure 3).

The WebGUI interface is based on the hBrowse[5] framework and, as such, affords the user significant scope to customise their job repository view. For example, jobs can be sorted by any of the available attributes (including identification number, job/subjob status, number of subjobs, number of submitted/running/failed subjobs etc). Some of these columns can also contain hyperlinks to further information, for example, when associated with a job that has subjobs, a link in the “ID” column (Figure 3) directs the user to a summary page showing detailed subjob information.

Additional functionality includes the ability for a user to quickly filter their jobs, for example, by displaying only failed subjobs or jobs that were executing a given application package. The job-repository view can also be configured to display only jobs created during a given time window, and the update frequency of the page changed to reflect the users needs (in discrete steps between 10 seconds and 20 minutes).

In addition to textual information, the WebGUI can display graphical data (generated using the Google Charts API[6]), such as the fraction of jobs in a particular state, or as a function of their backend or application type (Figure 4). In the event that a job has subjobs, information about these too can also be shown, including plots of the number of completed subjobs as a

![Figure 2. A user’s Ganga job repository, as displayed in the IPython interface.](image-url)
function of time (Figure 5) and pie charts illustrating the fraction of jobs in a given state, or destined for a particular backend.

4. The Ganga error-reporting tool

It is not uncommon for users to encounter problems when running complex analysis tasks, whether these be a result of poorly configured analysis packages, bugs in software code or problems at remote computing resources. Consequently, the larger research communities have invested effort in establishing user-support teams; optimisation of their valuable time is clearly essential to improve the service they provide, and help users focus on their analysis. Typically, such support is provided to the users by means of email groups (i.e. the user contacts a central support email address) or web-based ticketing systems (the user opens a support ticket online).

In order to fully debug a given problem, the support team quite often needs access to the files and/or computing environment in which the user was working, or where the software application executed. This, however, is usually very difficult or simply impossible as users are generally forbidden from passing their access credentials to third parties.

One obvious solution is for the user to directly send files (via email or a ticketing system) to the support team as requested. This places a burden on the user to correctly identify the necessary files, which can often be buried deep within a filesystem of which they have no prior knowledge. However, more transient information, such as the environment variables available to the executing code, still would not have been made available to the support team.

In an effort to provide a solution to this problem, the Ganga Error-Reporting Tool was developed. This exposes a Ganga function which, when called by the user (Figure 6), uploads the following job-specific information to a central server.

- Environment variables available to the Ganga session.
- Ganga-specific configuration parameters.
Figure 4. Global overview of a user’s Ganga job repository; pie charts show statistical job distribution as a function of their job status, backend and application type.

- A detailed log of the Ganga session messages.
- The Ganga client command-line history.
- Detailed book-keeping information relating to the job.
- All files required to execute the job (whether binaries, scripts or configuration files).

The receiving server also presents a frontend to the web (Figure 7). From where the user-support team can download the information in a single, compressed file archive, and analyse it to provide an appropriate response. Initially deployed as a generic Ganga component, the error-reporting tool has since been customised for the LHC CMS experiment and integrated into their main user analysis tool, CRAB, to provide a complete support-circle solution (schematically represented in Figure 8).

5. Conclusion
We have illustrated a selection of recent Ganga developments that were deployed with the common goal of improving the overall user experience and feel. The existing prepare method available to ATLAS users (Section 2) was extended to provide additional functionality and migrated into the core codebase, making it available to users of all application packages. In doing so we provided a mechanism which allows users to lock an application in a particular configuration (and preserve any associated files), with the knowledge that they can return to it at a later date and repeat the analysis task.
Figure 5. Global overview of a user’s Ganga job repository; pie charts show statistical job distribution as a function of their job status, backend and application type.

Figure 6. Example of the Ganga error-reporting tool invocation from the Ganga client.

We have also illustrated the Ganga WebGUI interface (Section 3), a development which compliments the traditional Ganga IPython client by providing a more powerful and customisable approach to viewing job repositories and their associated information.

Finally, we detailed the Ganga error-reporting tool (Section 4), an invaluable component in assisting community support groups provide an expedient solution to users’ problems, thereby
allowing them to focus on their analysis tasks. This tool has already been adopted by CMS, one of the major LHC experiments, and is integrated into their user analysis software package.

References
[1] Moscicki et al 2009 Ganga: a tool for computational-task management and easy access to Grid resources Computer Phys. Communications 110 pp2303-2316
[2] http://www.python.org
[3] Xiaomei Z et al 2012 J. Phys.: Conf. Series This volume
[4] Sciaba A, Ubeda Garcia M, Van Der Ster D C, Elmsheuser J, Medrano Llamas R, Legger F 2012 J. Phys.: Conf. Series This volume
[5] Koszkiewicz L et al 2012 Proc. EGI Community Forum 2012 / EMI Second Technical Conference 2012, PoS(EGICF12-EMITC2)062
[6] http://developers.google.com/chart/