

# Recent developments in user-job management with Ganga

Currie R<sup>1</sup>, Elmsheuser J<sup>2</sup>, Fay R<sup>3</sup>, Owen P H<sup>1</sup>, Richards A<sup>1</sup>, Slater M<sup>4</sup>, Sutcliffe W<sup>1</sup>, Williams M<sup>4</sup>

<sup>1</sup> Blackett Laboratory, Imperial College London, Prince Consort Road, London SW7 2BW UK

<sup>2</sup> Ludwig-Maximilians-Universität München, Elementary Particle Physics, Am Coulombwall 1, DE-85748 Garching, DE

<sup>3</sup> Department of Physics, University of Liverpool, Oliver Lodge, Oxford Street, Liverpool, L69 7ZE, UK

<sup>4</sup> Particle Physics Group, School of Physics and Astronomy, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK

E-mail: rcurrie@cern.ch

**Abstract.** The Ganga project was originally developed for use by LHC experiments and has been used extensively throughout Run1 in both LHCb and ATLAS. This document describes some the most recent developments within the Ganga project. There have been improvements in the handling of large scale computational tasks in the form of a new GangaTasks infrastructure. Improvements in file handling through using a new IGangaFile interface makes handling files largely transparent to the end user. In addition to this the performance and usability of Ganga have both been addressed through the development of a new queues system allows for parallel processing of job related tasks.

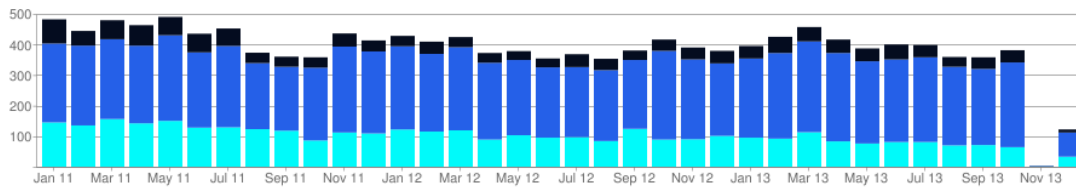
## 1. Introduction

Ganga [2] is an easy to use Job submission and management tool developed in Python [1]. The project has a large user-base with approximately 300 active daily users and is responsible for submitting and managing over 100,000 jobs per day. It has a proven track record and was heavily used by both the ATLAS and LHCb communities throughout Run1 of the LHC.

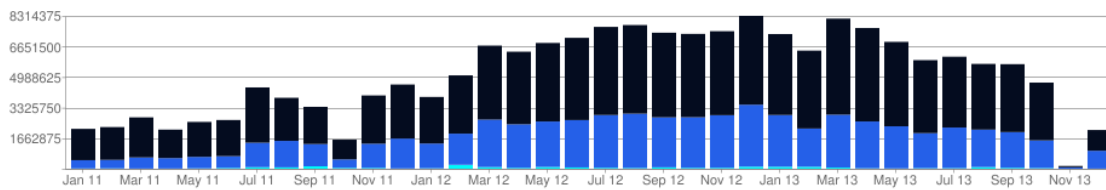
Ganga allows users to automate the tasks required to submit user programs varying from custom pyROOT applications to complex physics analyses developed around the Gaudi/Athena C++ frameworks. Job submission in Ganga is commonly performed through either a user written Ganga python scripts or an interactive Ganga environment making use of an IPython [4] console interface.

Some usage statistics collected from Ganga users throughout 2011 to 2013 are shown in Figures 1 and 2. Figure 1 shows the number evolution of the number of users throughout this time period showing the relative numbers of LHCb and Atlas users. Figure 2 shows the amount of jobs which were submitted during this time period.





**Figure 1.** This figure (in colour online) shows the number of unique Ganga users throughout the time period of 01/01/2011 to 31/12/2013. Dark blue indicates LHCb users, light blue indicates Atlas users and black indicates other users.



**Figure 2.** This figure (in colour online) shows the number submitted Ganga jobs submitted throughout the time period of 01/01/2011 to 31/12/2013. Black indicates the number of split jobs, dark blue the number of jobs which have been split and light blue the number of jobs which haven't been split.

## 2. Ganga Job Management

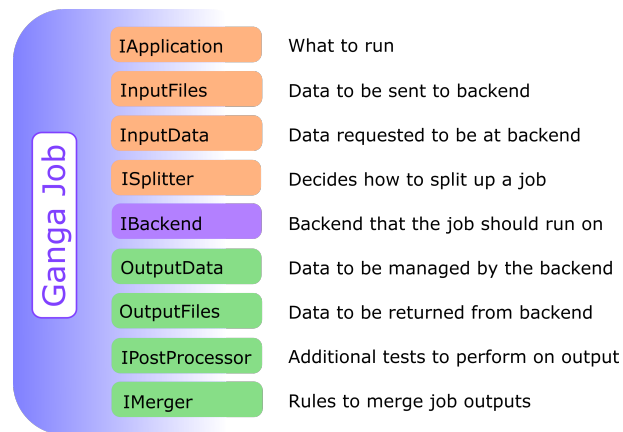
Ganga makes use of a set of plugin interfaces which allows for jobs to be described using a minimal job script. The configuration of these jobs is saved to disk using an XML based repository storing a jobs configuration and its current state.

A Ganga job is constructed using 9 possible plugins which can be used to describe how a job is to be structured. This is shown in Figure 3. These plugins make use the custom Ganga framework with objects inheriting from GangaObject class. These plugins allow Ganga to be able to package a chosen Application to be run and pass it to the processing backend. Using the inputfiles and inputdata objects Ganga is able to manage the data to be used by the Job. Additionally outputfiles and outputdata are used to manage any output data produced by the Job. Once a job has finished running Ganga is then able to automatically post-process and merge the results for the user.

To provide a simple and configurable job management system Ganga assigns each job a status. This is derived from the status of the job at the backend and allows for job flow management to be performed. This provides a generic way of managing jobs across multiple backends and provides the user with a generic interface for controlling their jobs.

## 3. IGangaFile Objects

Ganga 6 makes use of a common file interface named IGangaFile which allows for file objects to be easily managed and configured. Using this new interface Ganga is able to determine the most efficient way to transfer data between different storage solutions and the backend where the job is to be run. This allows users to construct jobs which run over a set of files without having to know implementation details of the storage solution. Ganga then automatically handles the steps required to make the data available to the job on the backend. Some of the most commonly used filetypes in Ganga 6 are described in Table 1.



**Figure 3.** This figure (in colour online) shows the components within a Ganga Job. The orange top 4 plugins are used to configure what is run and what input data is needed. The purple backend indicates where the job is to be run. The bottom 4 green plugins configure what is to be done once the job has completed.

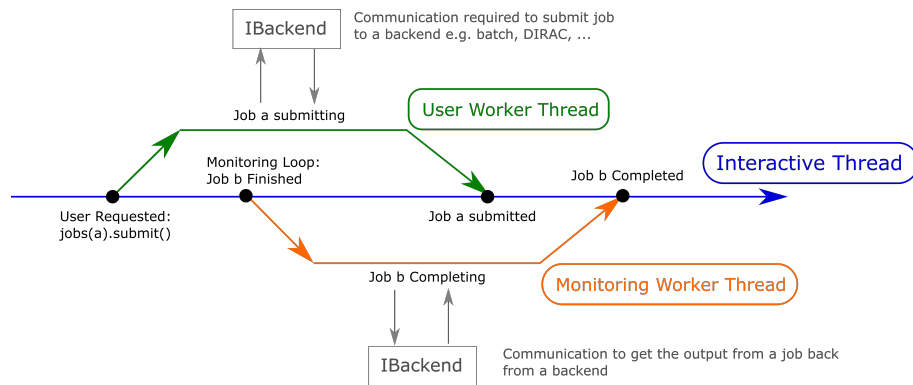
**Table 1.** A table describing some of the commonly used IGangaFile objects within Ganga 6.

File Type	Description
LocalFile	Files stored locally on disk accessible through common UNIX commands such as cp, rm etc..
DiracFile	Files stored on a Dirac managed Storage Element.
MassStorageFile	Files stored in a locally accessible storage system such as Castor [5] or EOS [6]
GoogleFile	Files stored on the Google Drive [7] service.
LCGSEFile	Files stored on an LCG Storage Element.
CernBoxFile	Files stored on the CERNBox [8] file storage system.

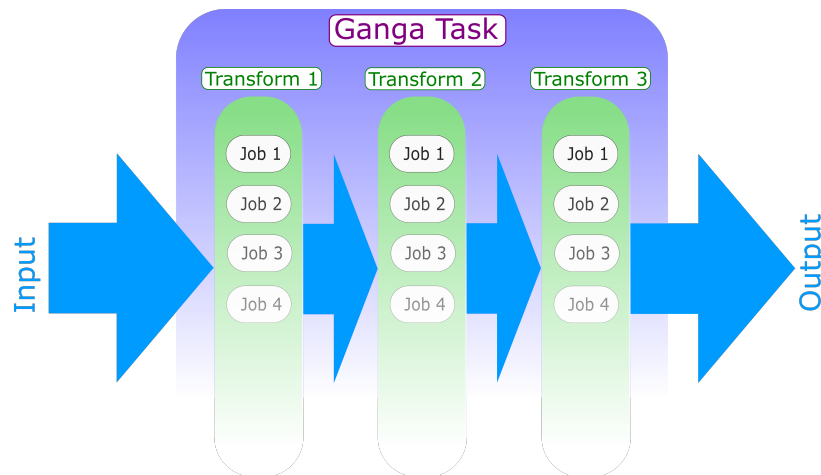
#### 4. Ganga Queues

Ganga 6 includes a new queues interface, accessible from the interactive Ganga prompt. This feature allows for time consuming methods to be completed in separate worker threads to the main interactive thread. A fixed number of worker threads are constructed on the launch of Ganga to perform User and Monitoring processing. This prevents the Ganga client from overloading the machine on which it's running. In order to make the most of the resources available, Ganga then queues additional processing requests until a Worker Thread instance becomes free. Figure 4 shows the new queues mechanism with worker threads. In this scenario the user is able to request for 'Job a' to be submitted to a backend making use of a User Worker Thread. This is then able to execute in a background Python thread. Whilst this is running the Monitoring Loop has determined that 'Job b' has completed and begins the task of completing the job in Ganga using a Monitoring Worker Thread.

Through the use of the queues interface, the users are able to register many different commands to be executed in the background away from the main Interactive Prompt. This allows for users to continue to manage other jobs and data whilst waiting for processes to complete.



**Figure 4.** Queuing system which runs additional computing tasks on either user worker threads or monitoring worker threads. This allows for a task such as `jobs(a).submit()` to be executed in a worker thread on the same machine without blocking the main interactive thread.



**Figure 5.** Illustration of how the tasks system within Ganga 6 allows for jobs to be automatically constructed based upon templates as required inputs become available.

## 5. Ganga Tasks

Ganga 6.1 introduces a new generic Tasks system to allow for a complex series of jobs to be chained together. This is achieved through the use of multiple transforms within a single Task. These tasks can make use of multiple transforms to chain jobs together in order to process a large computational request. Ganga Transforms are capable of mapping as one-to-one or one-to-many, making them useful for tasks such as data quality checking. Ganga Tasks are dynamic and capable of propagating data through various transforms as it becomes available. This is an improvement over an atomic design where one task cannot start until another has completed. This allows for the whole Task to be completed more efficiently. As well as this, Tasks are able to exploit many features within Ganga such as queuing and auto-resubmitting Jobs. Ganga provides some default preconfigured Tasks which allows for complex LHCb and ATLAS job chaining to be performed.

## 6. Conclusions

Ganga 6.0 has shown to be a stable platform on which to develop upon, with Ganga 6.1 taking advantage of this. It also has a proven track record of being a user-centric front-end to computing resources with many active users.

Looking toward Ganga 6.2 one of the areas of future development is an improved credential management system. One of the goals for future work is to allow for passing of additional user authentication information to the jobs running on their backends. This will allow for data to be seamlessly passed from a backend to a destination without requiring an instance of Ganga to intervene.

## References

- [1] Python Software Foundation. Python Language Reference, version 2.6. <http://www.python.org>
- [2] Moscicki J T et al, 2009 Ganga: a tool for computational-task management and easy access to Grid resources *Computer Physics Communications* Vol. **180** Issue 11
- [3] G. van Rossum and F.L. Drake Jr., 2006 The Python language reference manual: revised and updated for version 2.5 (Network Theory Limited, Bristol, 2006).
- [4] F. Perez and B.E. Granger, 2007 IPython: a system for interactive scientific computing , *Computing in Science and Engineering* **9-3** 21
- [5] CASTOR homepage <http://cern.ch/castor>
- [6] EOS homepage <http://eos.cern.ch>
- [7] Google Drive homepage <https://www.google.com/drive>
- [8] CERNBox homepage <http://cernbox.web.cern.ch>