GANGA: A GRID USER INTERFACE

K. Harrison, Cavendish Laboratory, University of Cambridge, CB3 0HE, UK
C.L. Tan, School of Physics and Astronomy, University of Birmingham, B15 2TT, UK
D. Liko, A. Maier, J.T. Moscicki, CERN, CH-1211 Geneva 23, Switzerland
U. Egede, Department of Physics, Imperial College London, SW7 2AZ, UK
R.W.L. Jones, Department of Physics, University of Lancaster, LA1 4YB, UK
A. Soroko, Department of Physics, University of Oxford, OX1 3RH, UK
G.N. Patrick, Rutherford Appleton Laboratory, Chilton, Didcot, OX11 0QX, UK

LHCb-PROC-2006-005
17/02/2006

Abstract

Details are presented of GANGA, the Grid user interface being developed to enable large-scale distributed analysis by physicists in the ATLAS and LHCb experiments.

INTRODUCTION

GANGA [1] is an easy-to-use frontend for job definition and management, implemented in Python [2]. It is being developed to meet the needs of ATLAS [3] and LHCb [4] for a Grid user interface, and is a key piece of the experiments' distributed-analysis systems [5, 6].

ATLAS and LHCb will investigate various aspects of particle production and decay in high-energy proton-proton interactions at the Large Hadron Collider (LHC) [7], due to start operation at the European Laboratory for Particle Physics (CERN), Geneva, in 2007. Both experiments will require processing of data volumes of the order of petabytes per year, and will rely on computing resources distributed across multiple locations. The experiments' data-processing applications, including simulation, reconstruction and physics analysis, are based on the GAUDI/ATHENA C++ framework [8]. This provides core services, such as message logging, data access, histogramming; and allows run-time configuration via options files. These can be written in Python, or using a value-assignment syntax similar to that of C++, and may have considerable complexity.

GAUDI/ATHENA jobs for simulation and reconstruction typically use software that results from a coordinated, experiment-wide effort, and is installed at many sites. The person submitting the jobs, possibly a production manager, performs the job configuration, which involves selecting the algorithms to be run, defining the algorithm properties and specifying inputs and outputs. The situation is similar for an analysis job, except that the physicists running a given analysis will usually want to load one or more algorithms that they have written themselves, and so use code that may be available only in an individual physicist's work area.

GANGA (GAUDI/ATHENA and Grid Alliance) deals with configuring the ATLAS and LHCb applications, allows switching between testing on a local batch system and large-scale processing on the Grid, and helps keep track of results. This paper outlines the system's design, and describes its use.

JOB REPRESENTATION

A job in GANGA is constructed from a set of building blocks (Fig. 1). All jobs must specify the software to be run (application) and the processing system (backend) to be used. Many jobs will specify an input dataset to be read and/or an output dataset to be produced. Optionally, a job may also define functions (splitters and mergers) for dividing a job into subjobs that can be processed in parallel, and for combining the resultant outputs.

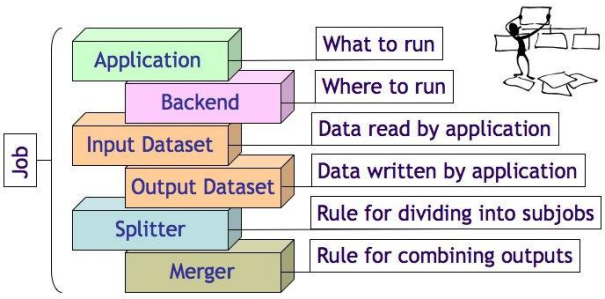

Figure 1: Building blocks for constructing a GANGA job.

Different types of application, backend, dataset, splitter and merger are implemented as plugin classes (Fig. 2). Each of these has its own schema, which places in evidence the configurable properties and their meanings. Properties are defined both to permit the user to set values defining the operations to be performed within a job, and to store information returned by the processing system, allowing tracking of job progress.

Plugins associated with a given category of job building block inherit from a common interface class - one of IApplication, IBackend, IDataset, ISplitter and IMerger. This documents the required methods – a backend plugin, for example must have submit and kill methods – and contains default (often dummy) implementations. The interface classes have a common base class, GangaObject, which provides a persistency mechanism and allows user default values for plugin properties to be set via a configuration file.

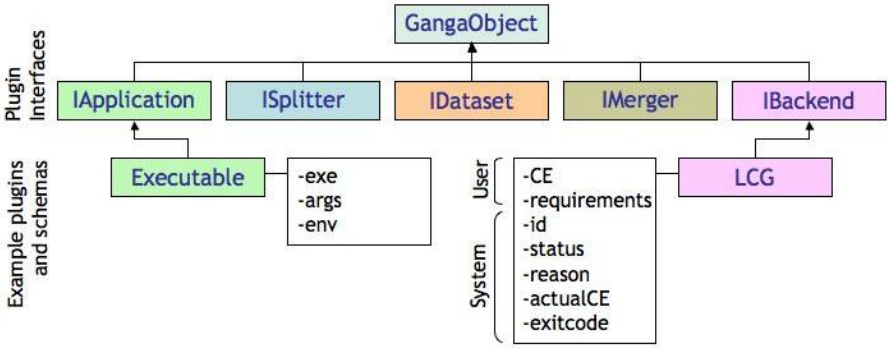

Figure 2: GANGA plugins. Plugins for different types of application, backend, dataset, splitter and merger inherit from interface classes, which have a common base class. Schemas for the Executable application and for the LCG backend are shown as examples.

Applications for which plugins have been written include a generic Executable application, the ATLAS ATHENA application, and the GAUDI-based applications of LHCb: GAUSS (simulation), BOOLE (digitisation), BRUNEL (reconstruction) and DAVINCI (analysis). The backend plugins cover generic distributed systems, such as the LHC Computing Grid (LCG) [9] and GLITE [10]; experiment-specific distributed systems, such as DIAL [11] in ATLAS and DIRAC [12] in LHCb; and local batch systems, including LSF, PBS and Condor. Other plugins provide for file-based datasets, for the splitting of these datasets, and for the merging of output histogram files. Users can easily add new plugin classes, or suppress the loading of unwanted classes. As a result, GANGA can readily be extended or customised to meet the requirements of different user communities.

ARCHITECTURE

The functionality of GANGA is divided between components (Fig. 3):

- the Application Manager deals with defining the task to be performed within a job, including the application to be run, the user code needed, the values to be assigned to any configurable parameters, and the data to be processed;
- the Job Manager takes care of any job splitting requested, packages up required user code, performs submission to the backend, monitors job progress, and retrieves output files when jobs complete;
- the Archivist provides a repository for storing job information, and manages Ganga's workspace, keeping track of the locations of input and output files;
- the Core deals with startup operations, mediates communication between the other components, and makes functionality available through the GANGA Public Interface (GPI);
- the Client allows access to GPI commands in any of three ways: through a shell – the Command Line Interface in Python (CLIP); using GPI scripts, or through a Graphical User Interface (GUI).

USER VIEW

Configuration

GANGA has default parameters and behaviour that can be redefined at startup using one or more configuration files, which use the syntax understood by the standard Python [2] ConfigParser module. Configuration files can be introduced at the level of system, group or user, with each successive file processed able to override settings from preceding files. The configuration files allow selection of the Python packages that should be initialised when GANGA starts, and consequently of the modules, classes, objects and functions that are made available through the GPI. They also allow modification of the default values to be used when creating objects from plugin classes, and permit actions such as choosing the log level for messaging, specifying the location of the job repository, and changing certain visual aspects of GANGA.

Command Line Interface in Python

GANGA's Command Line Interface in Python (CLIP) provides for interactive job definition and submission from an enhanced Python shell, IPython [13], with many nice features. A user needs to enter only a few commands to set application properties and submit a job to run the application on a chosen backend, and switching from one backend to another is trivial. CLIP includes possibilities for organising jobs in logical files, for creating job templates, and for exporting jobs to GPI scripts. Exported jobs can be freely edited, shared with others, and/or loaded back into GANGA. CLIP is especially useful for learning how GANGA works, for one-off job-submissions, and – particularly for developers – for understanding problems if anything goes wrong.

GPI scripts

GPI scripts allow sequences of commands to be executed in the GANGA environment, and are ideal for automating

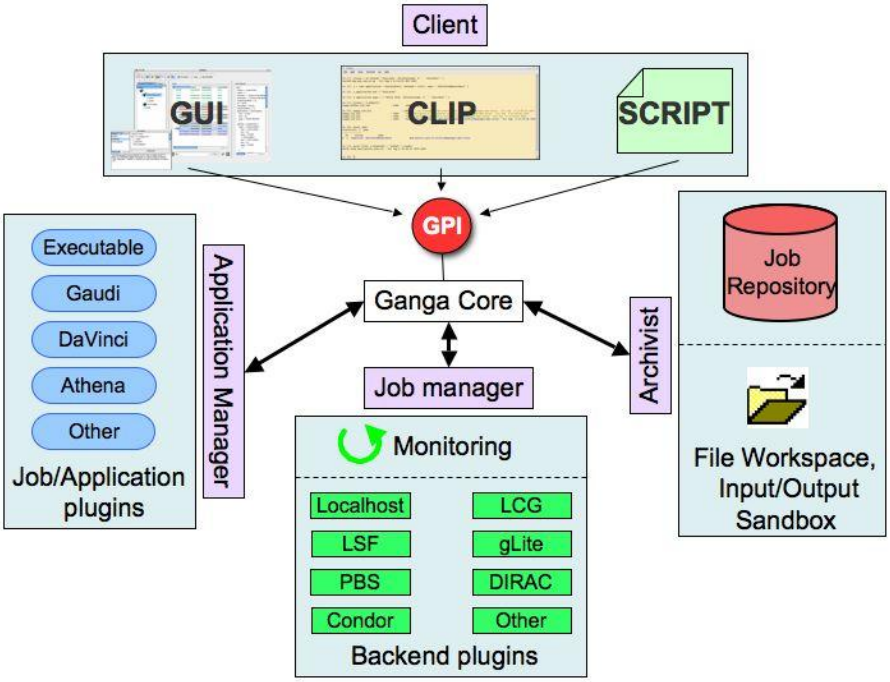

Figure 3: Schematic representation of the GANGA architecture. The main functionality is divided between Application Manager, Job Manager and Archivist, and is accessed by the Client through the GANGA Public Interface (GPI). The client can run the Graphical User Interface (GUI), the Command-Line Interface In Python (CLIP) or GPI scripts.

repetitive tasks.

GANGA includes commands that can be used outside of the Python/IPython environment to create GPI scripts containing job definitions; to perform job submission based on these scripts, or on scripts exported from CLIP; to query job progress; and to kill jobs. Working with these commands is similar to working with the commands typically encountered when running jobs on a local batch system, and for users can have the appeal of being immediately familiar.

Graphical User Interface

The GUI (Fig. 4) aims to further simplify user interaction with GANGA. It is based on the PyQt [14] graphics toolkit, and includes:

- a job tree, which allows browsing of logical folders;
- a monitoring panel, which shows the status of user jobs selected from the job tree;
- a job-details panel, which allows inspection of job definitions;
- a job-builder panel, for job creation and modification;
- a log panel, where message from GANGA are directed;
- a scriptor panel, which allows the user to execute arbitrary Python scripts and GPI commands, maximising flexibility when working inside the GUI.

By default, panels are shown together in a single window, but the panels are dockable, and can be resized and placed according to the tastes of the individual user. Job definition and submission is accomplished through mouse clicks and form completion, with overall functionality similar to CLIP.

Use within the experiments

Although still in the development phase, GANGA already has functionality that make it useful for physics studies. Tutorials for ATLAS and LHCb have been held in the UK and at CERN, and have led to GANGA being tried out by close to 100 people. Feedback has been positive, especially as regards the ease of use. GANGA has a small but growing number of frequent users, and is enabling them to run large-scale analyses successfully on the Grid, without needing to worry about Grid technicalities.

CONCLUSIONS

GANGA is being developed as a Grid user interface for physicists in the ATLAS and LHCb experiments. It simplifies configuration of applications based on the GAUDI/ATHENA framework used in these experiments; allows trivial switching between testing on a local batch system and running full-scale analyses on the Grid, hiding Grid technicalities; provides for job splitting and merging; and includes automated job monitoring and output retrieval. GANGA offers possibilities for working in an enhanced Python shell, with scripts, and through a graphical interface.

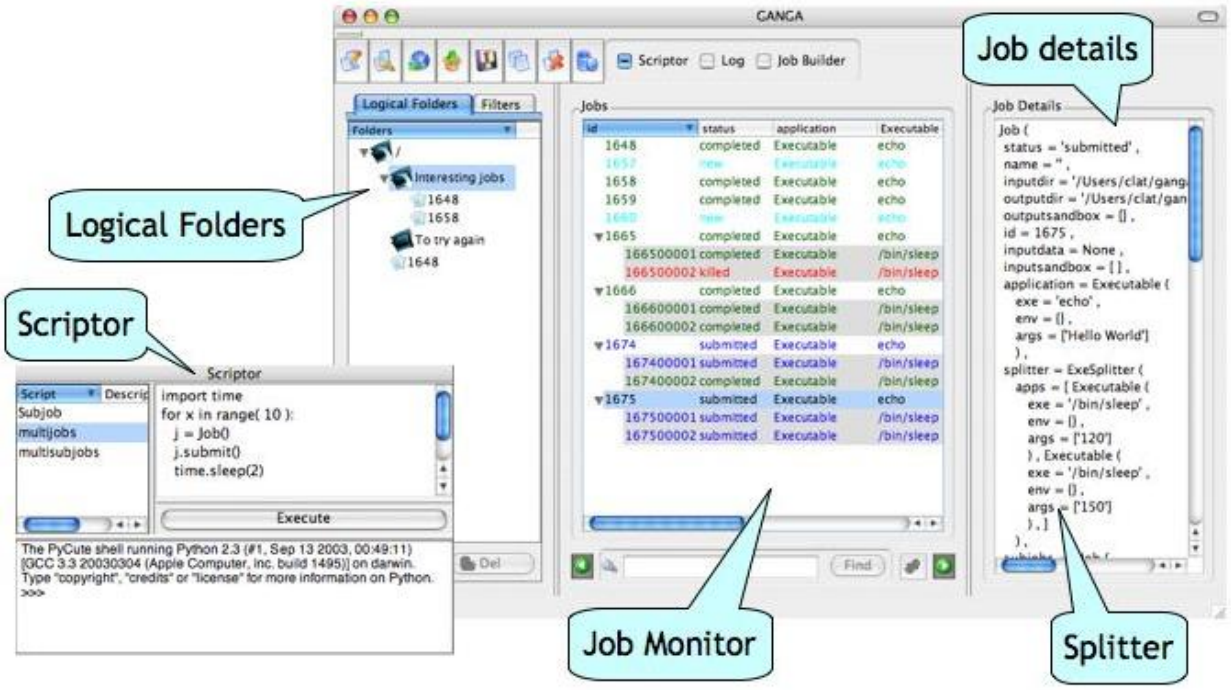

Figure 4: Screenshot of the GANGA GUI, showing: a main window (right), displaying job tree, monitoring panel and job details; and an undocked scriptor panel (left).

.

Although specifically addressing the needs of ATLAS and LHCb for running applications performing large-scale data processing on today's Grid systems, GANGA has a component architecture that readily allows extension to support other application types and future Grid evolutions. This choice of architecture, together with the possibilities allowed for configuration at startup, make GANGA potentially interesting for a range of user communities.

GANGA bas been tried out by close to 100 people from ATLAS and LHCb, and a small but growing number of physicists use GANGA routinely for running analyses on the Grid, with considerable success.

ACKNOWLEDGEMENTS

We are pleased to acknowledge support for the work on GANGA from GridPP in the UK and from the ARDA group at CERN. GridPP is funded by the UK Particle Physics and Astronomy Research Council (PPARC). ARDA is part of the EGEE project, funded by the European Union under contract number INFSO-RI-508833.

REFERENCES

[1] http://ganga.web.cern.ch/ganga/

[2] G. van Rossum and F. L. Drake, Jr. (eds.), Python Reference Manual, Release 2.4.3 (Python Software Foundation, 2006); http://www.python.org/

[3] ATLAS Collaboration, Atlas - Technical Proposal, CERN/LHCC94-43 (1994); http://atlas.web.cern.ch/Atlas/

[4] LHCb Collaboration, LHCb - Technical Proposal, CERN/LHCC98-4 (1998); http://lhcb.web.cern.ch/lhcb/

[5] D. Liko *et* al., The ATLAS strategy for Distributed Analysis in several Grid infrastructures, in: Proc. 2006 Conference for Computing in High Energy and Nuclear Physics, (Mumbai, India, 2006)

[6] U. Egede *et* al., Experience with distributed analysis in LHCb, in: Proc. 2006 Conference for Computing in High Energy and Nuclear Physics, (Mumbai, India, 2006)

[7] LHC Study Group, The LHC conceptual design report, CERN/AC/95-05 (1995); http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/

[8] P. Mato, GAUDI - Architecture design document, LCHb-98-064 (1998); http://proj-gaudi.web.cern.ch/proj-gaudi/welcome.html http://atlas-computing.web.cern.ch/atlas-computing/packages/athenaCore.php

[9] http://lcg.web.cern.ch/lcg/

[10] http://glite.web.cern.ch/glite/

[11] D. Adams *et* al., DIAL: Distributed Interactive Analysis of Large Datasets, in: Proc. 2006 Conference for Computing in High Energy and Nuclear Physics, (Mumbai, India, 2006); http://www.usatlas.bnl.gov/~dladams/dial/

[12] A. Tsaregorodtsev *et* al., DIRAC – the LHCb Data Production and Distributed Analysis system, in: Proc. 2006 Conference for Computing in High Energy and Nuclear Physics, (Mumbai, India, 2006)

[13] http://ipython.scipy.org/

[14] http://www.riverbankcomputing.co.uk/pyqt/