

# GANGA: A Grid User Interface for Distributed Data Analysis

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

## Abstract

Details are presented of GANGA, the Grid user interface being developed to enable large-scale distributed data analysis within High Energy Physics. In contrast to the standard LCG Grid user interface it makes transparent most of the Grid technicalities. GANGA can also be used as a frontend for smaller batch systems thus providing a homogeneous environment for the data analysis on inhomogeneous resources. As a clear software framework GANGA offers easy possibilities for extension and customization. We report on current GANGA deployment, and show that by creating additional pluggable modules its functionality can be extended to fit the needs of other grid user communities.

## 1. 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 a Grid user interface within the ATLAS [3] and LHCb [4] experiments in High Energy Physics, and is a key piece of their 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.

GANGA 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. All this information is contained within an object called GANGA job.

Jobs for simulation and reconstruction typically use GAUDI/ATHENA 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. Another major difference between analysis and production jobs consists in the amount of input data they process. As a rule an analysis job requires gigabytes or even terabytes of data collected in so called datasets and distributed among many storage elements around the globe. Discovery of dataset locations is done through recourse to various metadata and file catalogues and hence a mechanism for data discovery has to be included as an integral part of performing an analysis.

In Section 2 we give detailed description of a GANGA job. Section 3 describes the overall architecture of GANGA including the persistency and plugin systems. GANGA functionality and different aspects of user interface are summarized in Section 4. In Section 5 we report on current GANGA use within ATLAS and LHCb, and finally in Section 6 we provide more details about GANGA as a software framework and show ways how it can be extended to satisfy needs of other potential users.

## 2. 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. In this case after splitting the job becomes a master job and provides a single point of access for all its subjobs.

Different types of application, backend, dataset, splitter and merger are implemented as plugin classes (see Section 6). Each of these has its own schema, which describes 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.

Applications for which plugins have been written include a generic Executable application, the ATLAS ATHENA application, and the GAUDI-based applications of LHCb.

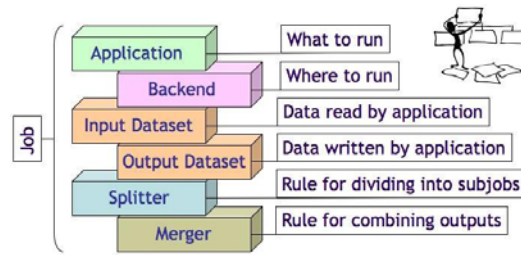


Figure 1: Building blocks for constructing a GANGA job.

The backend plugins cover generic distributed systems, such as the LHC Computing Grid (LCG) [9] and gLITE [10]; experiment-specific distributed systems, such as DIRAC [11] in LHCb; and local batch systems, including LSF, PBS and Condor.

## 3. Architecture

The functionality of GANGA is divided between components (Fig.2). GANGA Core links them together and performs most common tasks. It is represented by Application and Job Managers, Job Repository, and File Workspace. All components communicate via the GANGA Public Interface (GPI), which is designed to help GANGA users and external developers who are not proficient in Python to write their scripts and plugin modules more easily. 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).

### 3.1 Application Manager

The Application Manager deals with defining the task to be performed within a job, including the application to be run, the user code to be executed, the values to be assigned to any configurable parameters, and the data to be processed. GANGA calls the Application Manager for every job before its submission. The duty of the Application Manager is to return a configuration object which contains backend-independent information. In case of split jobs the operation of the Application Manager is optimised, so that it performs only those configuration actions for subjobs, which are not factorized in the master job. For jobs with a complicated configuration this can speed up the time required for job submission by orders of magnitude.

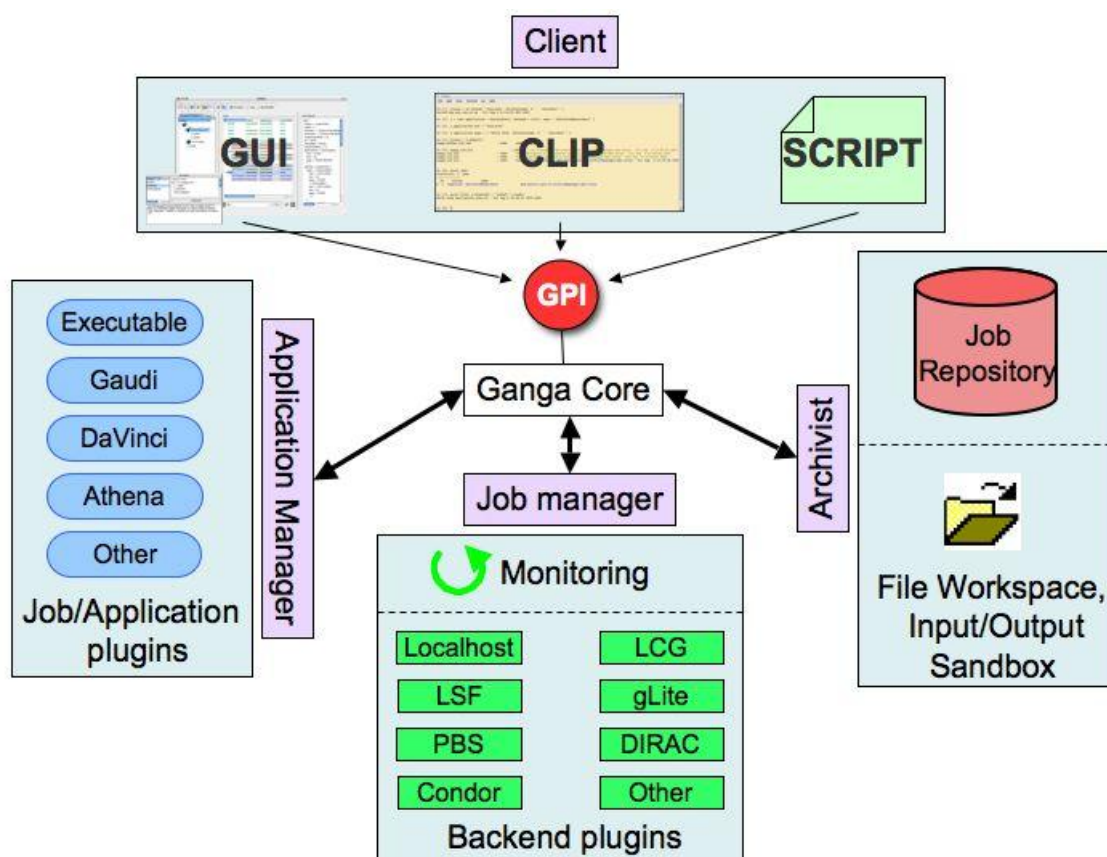


Figure 2: Schematic representation of the GANGA architecture. The main functionality is divided between Application Manager, Job Manager and Archivist, and is accessed by the Client through the GANGA Public Interface (GPI). The client can run the Graphical User Interface (GUI), the Command-Line Interface In Python (CLIP) or GPI scripts.

### 3.2 Job Manager

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. It takes as input the configuration object prepared by the Application Manager and calls the application runtime handler – a class which is responsible for the backend-specific part of the application configuration. Such a configuration mechanism reduces the number of required software modules because application runtime handlers are normally compatible with more than one type of application. It also allows dynamic connection to a new backend, and running the same application on different Grid or batch systems.

The Job Manager acquires information about job status with the help of the Monitoring

component that in turn polls plugin modules representing backends to which jobs were submitted.

### 3.3 Job Repository

Acting as a simple bookkeeping system the Job Repository stores all GANGA jobs. The Job Repository and File Workspace provide persistency for jobs and their input/output sandboxes so that GANGA can be restarted having access to the same information. If necessary this information can be searched for particular jobs according to job metadata which is also stored in the Job Repository.

The Job Repository can be either local or remote. Both types of repository have the same interface, so there is no difference from the programmatic point of view. However they differ substantially in implementation and in their usage.

The local Job Repository is a lightweight database written entirely in Python so that it can be easily run on any computer platform. It is mainly designed for individual users and provides possibility to work with GANGA off-line, for example prepare jobs for submission until on-line resources become available. In contrast, the remote Job Repository is a service centrally maintained for the need of many users. It gives the advantage of job access from any location where internet connection is present. Current implementation of the remote Job Repository is based on the AMGA metadata interface [12]. It supports secure connection and user authentication and authorization based on Grid proxy certificates.

The performance tests of both the local and remote repositories show good scalability for up to 10 thousand jobs per user, with the average time of job creation being 0.2 and 0.4 second for the local and remote repository correspondingly.

### 3.4 Plugin modules

GANGA comes with a variety of plugin modules representing different types of applications and backends. These modules are controlled correspondingly by the Application and Job Manager. Such a modular design allows the functionality to be extended in an easy way to suit particular needs as described in Section 6.

## 4. User view

User interacts with GANGA through the Client and configuration files.

### 4.1 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.

### 4.2 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. The 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. A realistic scenario of full submission of an analysis job in LHCb is illustrated in Fig 3.

### 4.3 GPI scripts

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

### 4.4 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 GPI, and consists of a set of dockable windows and a central job monitoring panel. The main window includes:

- a toolbar, which simplifies access to all the GUI functionality;
- a logical folders organiser implemented in a form of job tree;
- a job monitoring table, which shows the status of user jobs selected from the job tree;
- a job-details panel, which allows inspection of job definitions.

The Job Builder window has a set of standard tool buttons for job creation and modification. It also has a button for dynamic export of plugin methods, job attribute value entry widgets, and a job attribute tree view. The scriptor window allows the user to execute arbitrary Python

```

# Define an application object
dv = DaVinci(version = 'v12r15',
             cmt_user_path = '~/public/cmt',
             optsfile = 'myopts.opts')

# Define a dataset
dataLFN = LHCbDataset(files=[
'LFN:/lhcb/production/DC04/v2/00980000/DST/Presel_00980000_00001212.dst' ,
:
'LFN:/lhcb/production/DC04/v2/00980000/DST/Presel_00980000_00001248.dst'])

# Put application, backend, dataset and splitting strategy together in a job
# Dirac is the name of the LHCb submission system to the Grid.
j = Job(application=dv,
         backend=Dirac(),
         inputdata=dataLFN,
         splitter=SplitByFiles())

# Submit your complete analysis to the Grid.
j.submit()

```

Figure 3: A illustration of the set of CLIP commands for submission of a job to perform analysis in LHCb. While definition of the application includes elements specific to the LHCb experiment, the definition of the job follows the same structure everywhere.

scripts and GPI commands, maximising flexibility when working inside the GUI. Message from GANGA are directed to a log panel.

By default, all windows are shown together in a single frame, but because they are dockable they 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

## 5. Deployment

Although still in the development phase, GANGA already has functionality that makes 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 a 100 people.

Within LHCb Ganga has seen regular use for distributed analysis since the end of 2005. There are about 30 users at present of the system and performance has been quite satisfactory. In a test an analysis which took about 1s per event ran over 5 million events. For the analysis on the Grid the job was split into 500 subjobs. It was observed that 95% of the results had returned within less than 4 hours while the

remaining 5% initially failed due to a problem at a CE. These jobs were automatically restarted and the full result was available within 10 hours. The fraction of the analysis completed as a function of time is illustrated in Fig. 5. Recently more than 30 million events have been processed with the success rate exceeding 94%. Further details on the specific use of Ganga within LHCb can be found in [15].

In the last months GANGA found a new application outside the physics community. In particular, it was used for job submission to the EGEE and associated computing grids within the activities towards the avian flu drug discovery [16], and for obtaining optimum planning of the available frequency spectrum for the International Telecommunication Union [17].

## 6. Extensions

Despite similarities between ATLAS and LHCb they have some specific requirements and use cases for the Grid interface. In order to deal with these complications GANGA was designed as a software framework having a pluggable component structure from the very beginning. Due to this architecture GANGA readily allows extension to support other application types and

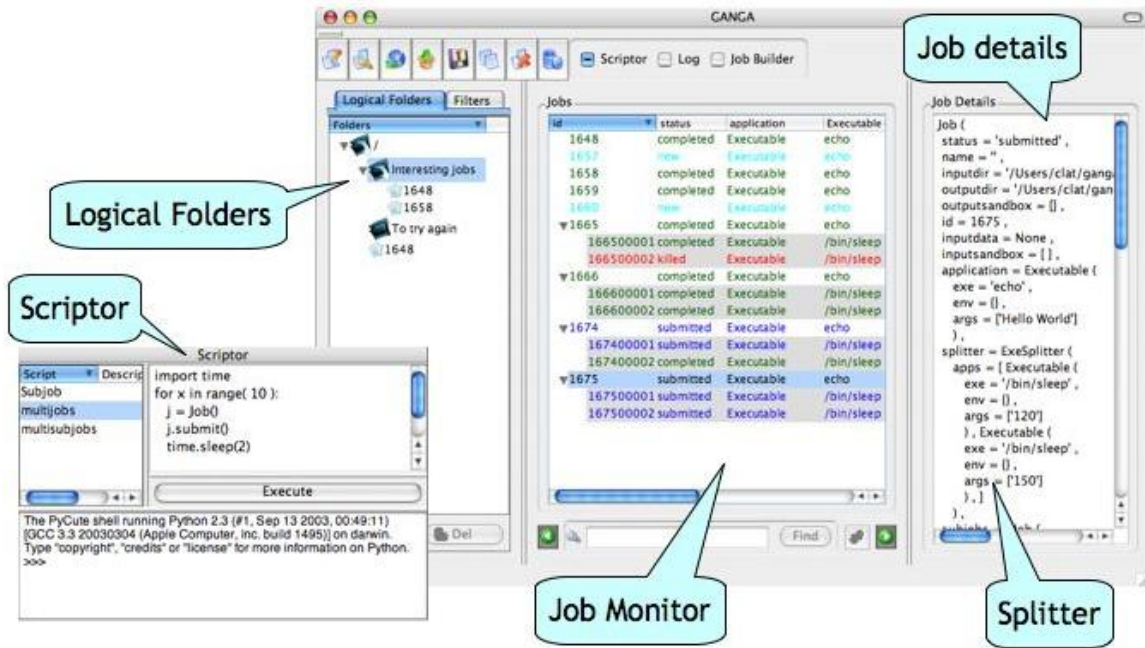


Figure 4: Screenshot of the GANCA GUI, showing: a main window (right), displaying job tree, monitoring panel and job details; and an undocked scriptor window (left).

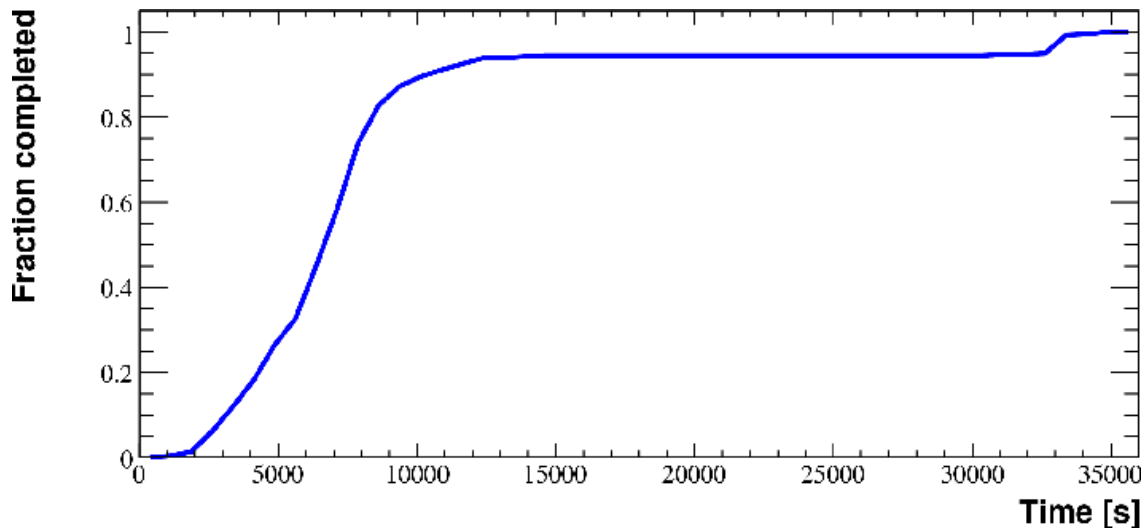


Figure 5: The fraction of the data analysis completed as a function of time

backends that can be found outside these two experiments. In particular, GANCA could be very useful for any computing task to be executed in a Grid environment that depends on sequential analysis of large datasets.

In practice plugins associated with a given category of job building block inherit from a common interface class - one of IApplication, IBackend, IDataset, ISplitter and IMerger as

illustrated in Fig. 6. The interface class documents the required methods -- a backend plugin, for example must have submit and kill methods -- and contains default (often empty) 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

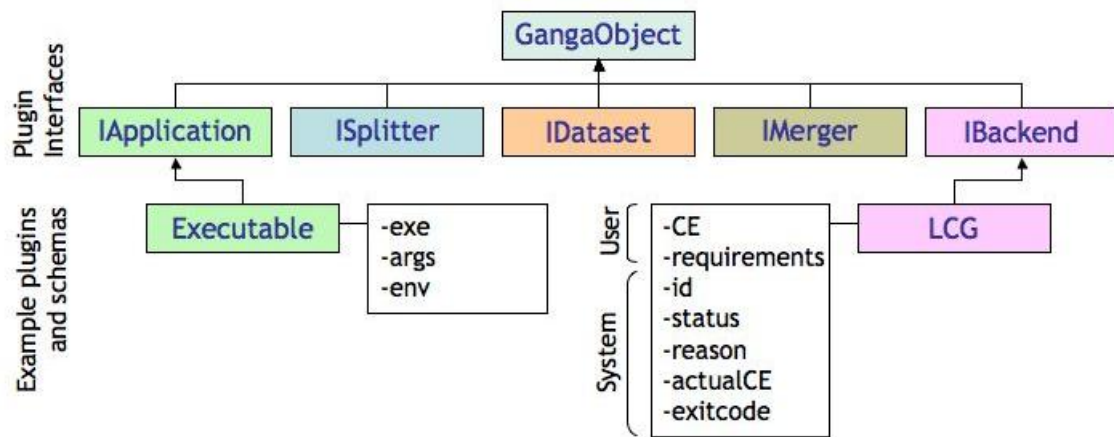


Figure 6: GANGA plugins. Plugins for different types of application, backend, dataset, splitter and merger inherit from interface classes, which have a common base class. Schemas for the Executable application and for the LCG backend are shown as examples.

file. Another function of the GangaObject class is to introduce a uniform description of all methods and data members visible to the user. Such a description called GANGA Schema allows GUI and CLIP representations of the plugin classes to be build automatically. Thus a GANGA developer can add a new plugin without special knowledge of the GUI and CLIP frameworks.

## 7. Conclusions

GANGA is being developed as a Grid user interface for distributed data analysis within the ATLAS and LHCb experiments. In contrast to many existing web portals [18] providing access to the Grid GANGA is a set of user tools running locally and therefore can offer more enhanced functionality. For example, it simplifies configuration of applications based on the GAUDI/ATHENA framework used in ATLAS and LHCb; 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.

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 can be easily adjusted for future Grid evolutions. As compared with similar Grid user interfaces [19] developed within other High Energy Physics experiments

GANGA has a clear application programming interface that can be exploited as an “engine” for implementing application-specific portals in different science areas. Written in Python GANGA can also benefit from more simple integration of new scientific applications as compared with the Java based Grid user interfaces such as Espresso [20].

GANGA has been tried out by close to 100 people from ATLAS and LHCb, and growing number of physicists use it routinely for running analyses on the Grid, with considerable success. Expansion of GANGA outside the High Energy Physics community was reported recently.

## 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://lhcb-new-homepage.web.cern.ch/lhcb-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] 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)
- [12] <http://project-arda-dev.web.cern.ch/project-arda-dev/metadata/>
- [13] <http://ipython.scipy.org>
- [14] <http://www.riverbankcomputing.co.uk/pyqt>
- [15] <http://ganga.web.cern.ch/ganga/user/html/LHCb/>
- [16] [http://www.eu-egee.org/press\\_releases](http://www.eu-egee.org/press_releases)
- [17] <http://indico.cern.ch/contributionDisplay.py?contribId=34&sessionId=11&confId=286>
- [18] <http://panda.physics.gla.ac.uk/>; <https://genius.ct.infn.it/>; <https://grid.ucla.edu:9443/gridsphere/gridsphere>
- [19] <http://cmsdoc.cern.ch/cms/ccs/wm/www/Crab/>; <http://alien.cern.ch/twiki/bin/view/AliEn/Home>; <http://projects.fnal.gov/samgrid/>
- [20] <http://freshmeat.net/projects/gridespresso>