User analysis of LHC*b* data with Ganga

Andrew Maier[1], Frederic Brochu[2], Greg Cowan[3], Ulrik Egede[4], Johannes Elmsheuser[5] , Benjamin Gaidioz[1], Karl Harrison[6], Hurng-Chun Lee[7], Dietrich Liko[8], Jakub Moscicki[1], Adrian Muraru[1], Katarina Pajchel[9], Will Reece[4], Bjørn Samset[9], Mark Slater[6], Alexander Soroko[10], Daniel van der Ster[1], Mike Williams[4], Chun Lik Tan[6]

[1] European Organization for Nuclear Research, CERN CH-1211, Genève 23, Switzerland
[2] Department of Physics, University of Cambridge, JJ Thomson Avenue, Cambridge CB3 0HE, United Kingdom
[3] Particle Physics Experiments Group, School of Physics, University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom
[4] Department of Physics, Imperial College London, Prince Consort Road, London SW7 2AZ, United Kingdom
[5] Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539 Munich, Germany
[6] School of Physics and Astronomy, The University of Birmingham, Edgbaston, Birmingham, B15 2TT, United Kingdom
[7] Nationaal instituut voor subatomaire fysica (NIKHEF), Science Park 105, 1098 XG Amsterdam, The Netherlands
[8] Institute of High Energy Physics of the Austrian Academy of Sciences, Nikolsdorfer Gasse 18, A-1050 Wien, Austria
[9] Experimental Particle Physics Group, Department of Physics, University of Oslo, PO Box 1048, Blindern, NO-0316 Oslo, Norway
[10] University of Oxford

E-mail: `andrew.maier@cern.ch`

Abstract. GANGA (`http://cern.ch/ganga`) is a job-management tool that offers a simple, efficient and consistent user analysis tool in a variety of heterogeneous environments: from local clusters to global Grid systems. Experiment specific plug-ins allow GANGA to be customised for each experiment. For LHC*b* users GANGA is the officially supported and advertised tool for job submission to the Grid. The LHC*b* specific plug-ins allow support for end-to-end analysis helping the user to perform his complete analysis with the help of GANGA. This starts with the support for data selection, where a user can select data sets from the LHC*b* Bookkeeping system. Next comes the set up for large analysis jobs: with tailored plug-ins for the LHC*b* core software, jobs can be managed by the splitting of these analysis jobs with the subsequent merging of the resulting files. Furthermore, GANGA offers support for Toy Monte-Carlos to help the user tune their analysis. In addition to describing the GANGA architecture, typical usage patterns within LHC*b* and experience with the updated LHC*b* DIRAC workload management system are presented.

1. Introduction

Submitting jobs to the Grid is a non-trivial task for the average non-technical user. It requires the user to:

- write a wrapper script to set up his job
- manage the in- and outputs of the job
- write a JDL to steer the jobs
- monitor the progress, success or failure of his jobs

At the same time, the Grid may not be the only computational resource available to the user. He may have access to a large local batch farm, or he might simply want to run his job locally on his desktop machine for testing purposes. In all these cases, the way to run a job is slightly different and requires the user to learn a multitude of commands and techniques instead of being able to concentrate on the analysis he wants to perform.

Ganga [1] provides a solution for the user to address all these issues. Starting off as common project of the ATLAS and LHC*b* collaborations and written in Python [3] it is freely available under the GNU Public License [4]. It is an easy to use front-end for the submission of computing jobs to a variety of computing resources such as the local machine, a variety of batch systems and various computing Grid flavours. The Ganga philosophy (see Figure [1]) is to try to make the job submission to different submission backends as transparent as possible.

Ganga includes a plug-in system which allows it to be extended both for submission backends and application plug-ins. While submission backends extend the possibility of where to send a job, application plug-ins allow simplified job submission of applications commonly used within a user community . Currently Ganga includes submission backends for batch systems, such as LSF, PBS, SGE and Condor [5], Grid systems such as LCG [6] and NorduGrid [7] and workload

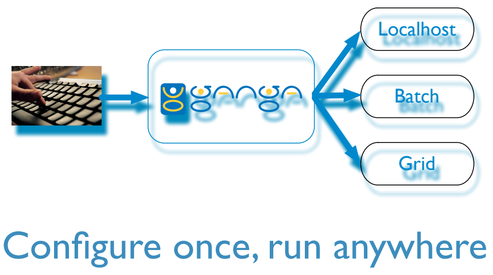


Figure 1. The basic concept of Ganga. Ganga attempts to be as transparent as possible when submitting a job to different submission backends. A job sent to the Grid should look no different than a job sent to the local machine.

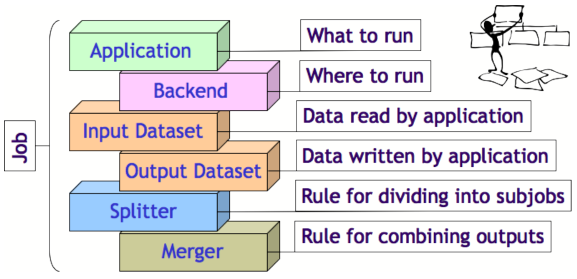


Figure 2. The concept of a Ganga job. Users are required to specify at least an application to run and a backend, where to run.

management systems (WMS), typically sitting on top of Grid middleware such as the PanDA [8] system by ATLAS and the DIRAC [9] system by LHC*b*. In addition Ganga is shipped with customised application handers, such as the Root [10] handler and the experiment specific application handlers for Athena (ATLAS) and Gaudi (LHC*b*). A general catchall executable application handler exists, capable of handling any application. However, specialised application handlers can offload a lot of the work needed to write special wrapper scripts and to configure the application separately. This simplification is achieved by knowing how the application has to be run, how to set up the necessary environment and which inputs and outputs are to be expected.

This paper will describe the LHC*b* specific plug-ins to Ganga which include the Gaudi [2] application plug-ins as well as the DIRAC backend plug-ins specific to LHC*b*. Additional information on Ganga in general can be found in [11], whereas the ATLAS specific plug-ins is covered in [12] In addition, a typical workflow for an LHC*b* user performing an analysis is presented. Finally, some user statistics for LHC*b* are presented, showing the widespread acceptance within LHC*b*.

2. Ganga plug-ins for LHC*b*

2.1. The Gaudi *plug-in*

The LHC*b* user prepares and writes his own `C++` code, normally called an *Algorithm* to be loaded into the running application as shared libraries. In addition a python file called an *options file* is used to steer the application, load additional algorithms, transmit values to algorithms and declare which input and output files are used.

In order to to run his job a user has to set up a runtime environment by using the code management tool CMT. Once the environment is set up the user can run the application by

```
Out[3]: DaVinci (
    extraopts = None ,
    package = 'Phys' ,
    configured = None ,
    masterpackage = None ,
    platform = 'slc4_ia32_gcc34' ,
    version = 'v22r1' ,
    setupProjectOptions = '' ,
    user_release_area = '~/cmtuser' ,
    optsfile = [File(name='~/cmtuser/myopts.py',subdir='.')]
 )
```

Figure 3. Example of a DaVinci application object. Besides specifying the version to be used, the main parameter is the `optsfile` parameter.

```
j.splitter=DiracSplitter(filePerJob=20)
print j.splitter
DiracSplitter (
    filesPerJob = 20 ,
    maxFiles = -1 ,
    ignoremissing = False
    )
```

Figure 4. Example of a `DiracSplitter` object. This allows splitting of large jobs into more manageable subjobs, thereby guaranteeing that the data for the subjob is available at the site.

```
 j.merger=RootMerger(files=['myhisto.root'])
print j.merger
RootMerger (
    files = ['histo.root'] ,
    ignorefailed = False ,
    overwrite = False
    )
```

Figure 5. A `RootMerger as an example of a merger object`. Mergers allow the results of subjobs to be merged back into common results for all subjobs of a master job.

specifying the application name with the options file name as an argument. The user will have to take care of setting up the environment, making sure that he knows where his output file will be located and how to retrieve it, e.g. also in the case of using the Grid. In addition, if the job is long, the user can profit from running several jobs in parallel. For this he then has to specify several job options files with different input data file statements. Finally, the user must check and validate all the running jobs himself.

GANGA allows the user to considerably simplify this task. Through a specialised application plug-in for LHC*b* applications, GANGA allows the user to simply specify the name and version of the application, the options file to be used and the location of the algorithm to be included (See Figure 3 as an example).

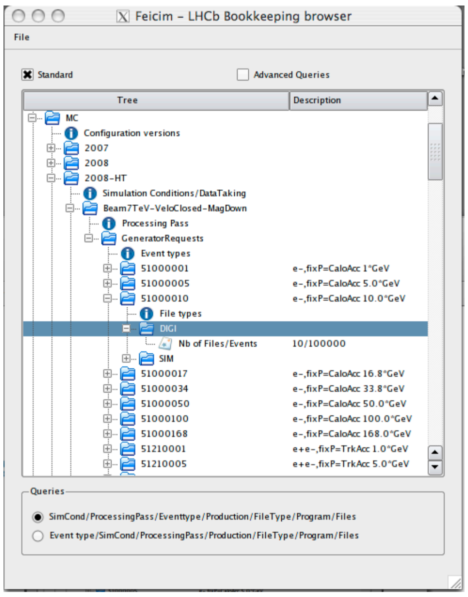
Since the GANGA GAUDI plug-in is aware of how to configure and run an LHC*b* application several tasks can be performed for the user.

```
LHCbDataset (
 cache_date = 'Wed Aug 29 23:49:04 2007' ,
 files = [ LHCbDataFile (
    name = 'LFN:/lhcb/production/[...]/DST/0000/00001889_00000003_5.dst' ,
    replicas = ['IN2P3-disk', 'CERN-disk']
    ) ,
        ...]
 )
```

Figure 6. An `LHCbDataset` object.

First of all GANGA, through the GAUDI plug-in, knows how to set up the environment for an LHC*b* job. Next, by parsing the options file, GANGA's GAUDI application plug-in can determine inputs and output files and can register these files as in- and outputs of part of the job. With the knowledge of the algorithm to be used, GANGA can determine the shared libraries to be copied to a different location, enabling the user to immediately continue his development work once he has submitted the job. This also allows to *replay* the job exactly as submitted at a later point, either for verification purposes, or more frequently to resubmit jobs that have crashed or did not run successfully at the chosen backend. Finally, since GANGA is aware of the input data to be used, it is able to split the job based on various criteria into a list of subjobs that are then submitted independently. The most common splitter used in LHC*b* is the `DiracSplitter` (see Figure 4) which splits the list of input files into sub-lists for each job. These sub-lists are created in such a way that each file within a sub-list contains at least one overlapping file replica on the LHC*b* Tier 1 sites. This guarantees that the job will be able to run over the specified data.

The splitter also allows to save space and resources, by only storing the differences between the jobs in the subjob sandbox, whereas the master job has all the sandbox files commonly used



```
Feicim – LHCb Bookkeeping browser
File

[✓ Standard]   Advanced Queries

Tree                                          Description
MC
  Configuration versions
  2007
  2008
  2008-HT
    Simulation Conditions/DataTaking
    Beam7TeV-VeloClosed-MagDown
      Processing Pass
      GeneratorRequests
        Event types
        51000001                              e-,fixP=CaloAcc 1*GeV
        51000005                              e-,fixP=CaloAcc 5.0*GeV
        51000010                              e-,fixP=CaloAcc 10.0*GeV
          File types
          DIGI
            Nb of Files/Events                10/100000
          SIM
        51000017                              e-,fixP=CaloAcc 16.8*GeV
        51000034                              e-,fixP=CaloAcc 33.8*GeV
        51000050                              e-,fixP=CaloAcc 50.0*GeV
        51000100                              e-,fixP=CaloAcc 100.0*GeV
        51000168                              e-,fixP=CaloAcc 168.0*GeV
        51210001                              e+e-,fixP=TrkAcc 1.0*GeV
        51210005                              e+e-,fixP=TrkAcc 5.0*GeV

Queries
  [✓ SimCond/ProcessingPass/Eventtype/Production/FileType/Program/Files]
  Event type/SimCond/ProcessingPass/Production/FileType/Program/Files
```

Figure 7. A screen-shot of the DIRAC Bookkeeping Browser integrated into Ganga. It allows the user to select the data used for analysis. The data can be directly stored in customised data-set objects

in all jobs.

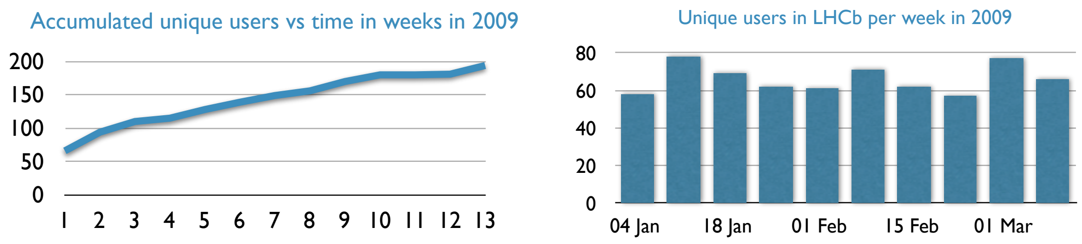


Figure 8. Accumulated number of unique LHCb users and unique users per week of Ganga in the first three months of 2009.

Finally a *Merger* (see Figure 5) allows to merge the results for the subjobs into one consistent result for the entire master job. The most commonly used merger for LHCb users is the `RootMerger`, which merges Root files, but other mergers such as a text merger exist.

2.2. The DIRAC plug-in

Another LHCb specific plug-in in Ganga is the DIRAC submission backend. It allows LHCb users to submit jobs to the Grid via the DIRAC WMS. The DIRAC backend adds a layer of redundancy and reliability onto the existing Grid middleware and allows the LHCb collaboration to fair-share its resources within the collaboration. From the user perspective, the DIRAC plug-in in Ganga is very simple and with the exception of the `CPUtime` parameter there are no user configurable settings. However, the DIRAC plug-in class has additional methods, which allow

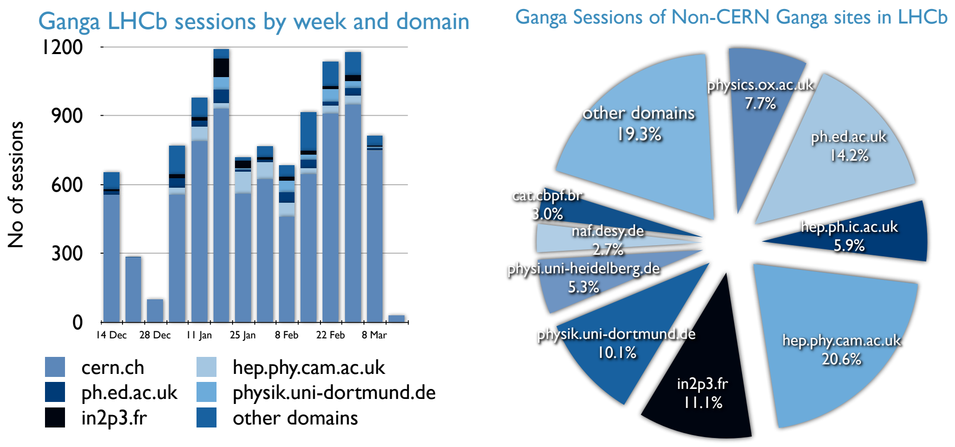
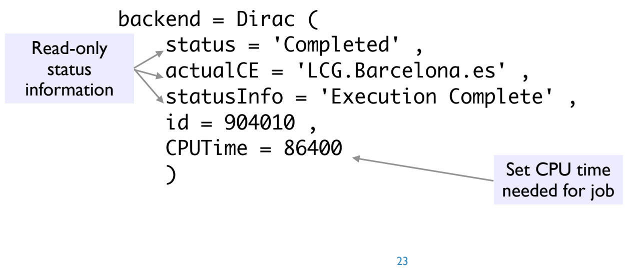


Figure 9. The distribution of LHC*b* GANGA sessions per week and domain (left) and the distributions of LHC*b* GANGA session by domain (excluding CERN).

the user to retrieve large outputs which cannot be transferred via the sandbox to be retrieved locally.



```
backend = Dirac (
    status = 'Completed' ,
    actualCE = 'LCG.Barcelona.es' ,
    statusInfo = 'Execution Complete' ,
    id = 904010 ,
    CPUTime = 86400
    )
```

Figure 10. The DIRAC backend object. All but the `CPUTime` item are read-only properties

Integrated into the new version of DIRAC is the LHC*b* bookkeeping system, a metadata catalogue, which allows the user to search for data to be analysed. The graphical user interface for this browser (see Figure 7) is integrated into GANGA and allows the user to search and directly insert the data into an `LCHbDataset` (see Figure 6). The `LCHbDataSet` allows the data to be specified independently from the GAUDI options file and can store additional information, such as a cache of the replicas of each file, used to split files using the `DiracSplitter`.

3. Ganga user experience in LHC*b*

The GANGA monitoring [13] attempts to monitor the usage pattern of GANGA users. It records statistics like e.g. the kind of loaded plug-ins, the type of session used, the GANGA version used, the domain of the user and the number of unique users.

Using results from this monitoring for LHC*b* a steadily increasing number of users are visible, currently reaching around 200 users in total (see Figure 8). This is larger than the expected number of LHC*b* users foreseen in the LHC*b* TDR [14] to be using the Grid. Also visible from Figure 8 is the fact that GANGA is used on a regular basis by a large fraction of the LHC*b* community with more than 60 users per week.

From Figure 9 one can see that a large number of users are submitting jobs from CERN, this can be explained by the fact that the GANGA developer team maintains a reference installation of GANGA accessible to all users on the public linux cluster `LXPLUS`. A number of external installations are also visible, with almost 50% of the users coming from the United Kingdom, followed by Germany and France. This success is certainly due to large number of tutorials given in addition to the tutorials given every three months in the LHC*b* software weeks.

4. Conclusions

GANGA is a well established and accepted user interface for job submission to batch and Grid systems. With more than 200 users it now reaches almost all analysis users in LHC*b*. In addition more than 60 LHC*b* users per week use GANGA which again proves its popularity. The specialised application plug-ins help all LHC*b* users to simply the submission, splitting and merging of large scale jobs and to manage them efficiently. A submission backend plug-in to the LHC*b* DIRAC system allows users to benefit from the redundancy and advanced reliability of this system.

References

[1] F. Brochou et al, Ganga – a tool for computational-task management and easy access to Grid resources, submitted to Computer Physics Communications (arXiv:0902.2685v1)
[2] Gaudi provides the necessary interfaces and services for building HEP experiment frameworks in the domain of event data processing applications. `http://www.cern.ch/gaudi`.
[3] The Python scripting language `http://www.python.org`
[4] The GNU General Public License, `http://www.gnu.org/licenses/gpl.html`.
[5] LSF `http://www.platform.com`, PBS `http://www.openpbs.org`, SGE `http://gridengine.sunsource.net`, Condor, D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience", **17, 2-4** (2005), 323, `http://www.cs.wisc.edu/condor/`
[6] Worldwide LHC Computing Grid, `http://www.cern.ch/LHCgrid`.
[7] M. Ellert *et al.*, "Advanced Resource Connector middleware for lightweight computational Grids", Future Generation Computer Systems (2007) **23**.
[8] T Maeno. PanDA: distributed production and distributed analysis system for ATLAS. 2008 J. Phys.: Conf. Ser. 119 062036 (4pp) doi: 10.1088/1742-6596/119/6/062036.
[9] A.Tsaregorodtsev *et al.*, "DIRAC, The LHCb Data Production and Distributed Analysis System", Proceedings from CHEP06, 2006.
[10] Rene Brun and Fons Rademakers, "ROOT - An Object Oriented Data Analysis Framework", Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nuclear Ins. Methods Phys. Res., **A389** (1997) 81. See also `http://root.cern.ch`.
[11] D C Vanderster et al. A PanDA Backend for the Ganga Analysis Interface. These proceedings.
[12] J Elmsheuser et al. Distributed Analysis in ATLAS using GANGA. These proceedings.
[13] Ganga Monitoring Pages `http://gangamon.cern.ch:8888/`
[14] Antunes-Nobrega, R *et al.* [LHCb Collaboration], "LHCb TDR computing technical design report", CERN-LHCC-2005-019