Journal of Physics: Conference Series **219** (2010) 072056 doi:10.1088/1742-6596/219/7/072056

A PanDA Backend for the Ganga Analysis Interface

D C Vanderster[1], J Elmsheuser[2], D Liko[3], T Maeno[4], P Nilsson[5], T Wenaus[4] and R Walker[2]

[1] European Organization for Nuclear Research, CERN CH-1211, Genève 23, Switzerland
[2] Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539 Munich, Germany
[3] Institute of High Energy Physics of the Austrian Academy of Sciences, Nikolsdorfer Gasse 18, A-1050 Wien, Austria
[4] Brookhaven National Lab, P.O. Box 5000, Upton, NY 11973-5000, USA
[5] University of Texas at Arlington, 701 South Nedderman Drive, Arlington, TX, 76019, USA

E-mail: daniel.colin.vanderster@cern.ch

Abstract. Ganga provides a uniform interface for running ATLAS user analyses on a number of local, batch, and grid backends. PanDA is a pilot-based production and distributed analysis system developed and used extensively by ATLAS. This work presents the implementation and usage experiences of a PanDA backend for Ganga. Built upon reusable application libraries from GangaAtlas and PanDA, the Ganga PanDA backend allows users to run their analyses on the worldwide PanDA resources, while providing the ability for users to develop simple or complex analysis workflows in Ganga. Further, the backend allows users to submit and manage "personal" PanDA pilots: these pilots run under the user's grid certificate and provide a secure alternative to shared pilot certificates while enabling the usage of local resource allocations.

1. Introduction

The ATLAS detector at CERN, scheduled to start taking data in fall 2009, will produce vast quantities of data for particle physicists to analyse. The ATLAS collaboration, in cooperation with a number of organizations worldwide, has deployed an international distributed computing infrastructure with resources sufficient to store and process the particle collision data. A significant component of this "grid" infrastructure has been developed to allow physicist users to execute arbitrary applications on data that is distributed geographically.

As a large collaboration with a few thousand members, ATLAS has developed a number of competing tools and technologies in the area of distributed analysis. The various analysis workflows at use in ATLAS are depicted in Figure 1 (dashed lines indicate connections which are in progress or under development). Two front-end tools are available to users: Ganga [1] and the PanDA Client [2, 3], which contains the tool *pathena*. Because each of these tools has a comparable user community, ATLAS has decided to continue support of the two front-ends. The two front-ends can access the resources of three grids: the Open Science Grid [4], the EGEE Grid [5], and NorduGrid [6].

This work presents a PanDA backend for Ganga. Built with reused components of the PanDA Client, this new backend is being developed to allow Ganga users to make use of the growing number of resources managed by the PanDA workload management system.

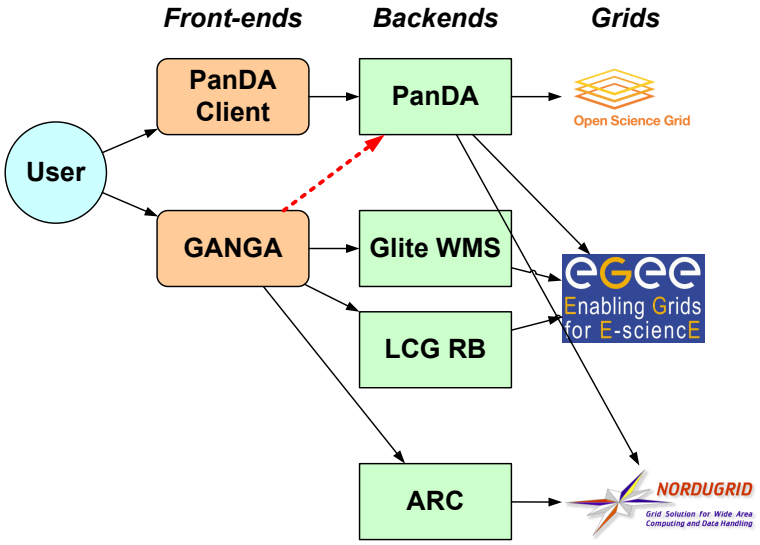

Figure 1. ATLAS Distributed Analysis Workflows.

The remainder of this paper is organized as follows. Section 2 presents an overview of the Ganga front-end tool and the PanDA workload management system. Section 3 presents the design and implementation of the PanDA backend for Ganga. Section 4 demonstrates a novel usage of Ganga to work around security limitations by delivering personal pilot jobs directly to grid resources. Finally, Section 5 concludes and describes the future directions.

2. Ganga and PanDA Overviews

This work provides an extension for the Ganga user analysis tool to access resources managed by PanDA. These are described here.

2.1. Overview of Ganga

Ganga [1] is a front-end user tool which is jointly developed by LHCb and ATLAS to allow users to run analyses on arbitrary resources (e.g. locally, on a batch system, or on the grid). It is an open source project and is used and extended by many communities outside of high-energy-physics. The Ganga philosophy is "configure once, run anywhere"; this model allows users to incrementally scale up their analyses from test jobs on a laptop to full scale analysis on the grid. Figure 2 depicts the key components of a Ganga job; ATLAS has modules developed to enable Athena analyses of the ATLAS datasets on all three of the ATLAS grids.

Ganga presents three different interfaces to the users: a command line interface, an interactive python interface, and a graphical user interface (GUI). Each of these interfaces has associated pros and cons The command line interface allows users to easily switch from local analysis with `athena <options>` to grid analysis simply by typing `ganga athena <options>`. The interactive python interface is the most commonly used by experienced users; it presents a powerful tool for the manipulation of job objects, including commands to select, configure, submit, copy, resubmit, kill, and remove jobs. Finally, the GUI can be used to manipulate jobs using a point-and-click interface; notably, large numbers of running jobs can be more easily monitored using the GUI.

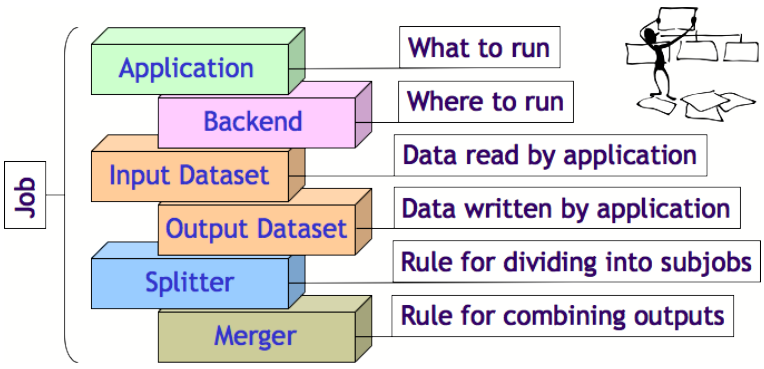

Figure 2. Ganga job is composed of 6 key modules.

2.2. Overview of PanDA

The Production ANd Distributed Analysis (PanDA) [2] system is a workload management system developed to meet the needs of ATLAS distributed computing. PanDA is built around the concept of pilot jobs; rather than sending jobs directly from a client to a worker node, pilot-based systems use an intermediate server to store the actual jobs, then make use of generic pilot jobs which retrieve actual jobs from the server. This model allows for increased flexibility in resource management, specifically that generic pilot jobs can be programmed to retrieve an arbitrary job from the queue, and bypasses the queuing delays seen at each hop of the traditional client-worker node model (i.e. at the central queue then the site queue). One trade-off in pilot based models is that the generic pilot jobs are often submitted by just a few operators, meaning that user jobs run under a different identity than the submitter. This can have implications for site-level usage accounting, data security, and site-level fair-share implementations.

PanDA has proven to be very successful in managing the ATLAS distributed production requirements, and is now being extended to manage distributed analysis on all three ATLAS grids. As such, backend support for PanDA in Ganga is important in order to provide users who familiar with Ganga with access to all ATLAS resources.

3. A PanDA Backend for Ganga

Starting with Ganga version 5[1], users have been able to submit Athena analyses on the PanDA system using the GangaPanda module. This module provides all the configuration, detection, and packaging of user analysis code to run on PanDA-managed worker nodes. The Ganga philosophy of "configure once, run anywhere" is demonstrated by allowing users to make a simple change to their job script to run on PanDA; namely, jobs configured with `backend=LCG()` or `backend=NG()` can be modified to run on PanDA using `backend=Panda()`.

The GangaPanda module is composed of two key components: the backend object (which handles submissions to Panda), and the runtime handlers (which map the Athena, AthenaMC, and Executable applications to the backend). The backend object handles all of the interactions with the PanDA server, namely submission, monitoring, cancellation, and resubmission of job objects. It also provides a class of functions that can be reused by the runtime handlers, including functions to perform resource brokerage, upload source files to the server, and retrieve performance statistics about running or completed jobs. The role of runtime handlers is to map Ganga applications (e.g. Executable, Athena, or AthenaMC) to the backend object by packaging the application in a format recognizable by the backend system.

[1] Ganga 5 has support for the Athena application on Panda; Ganga 5.2 adds support for Executable and AthenaMC.

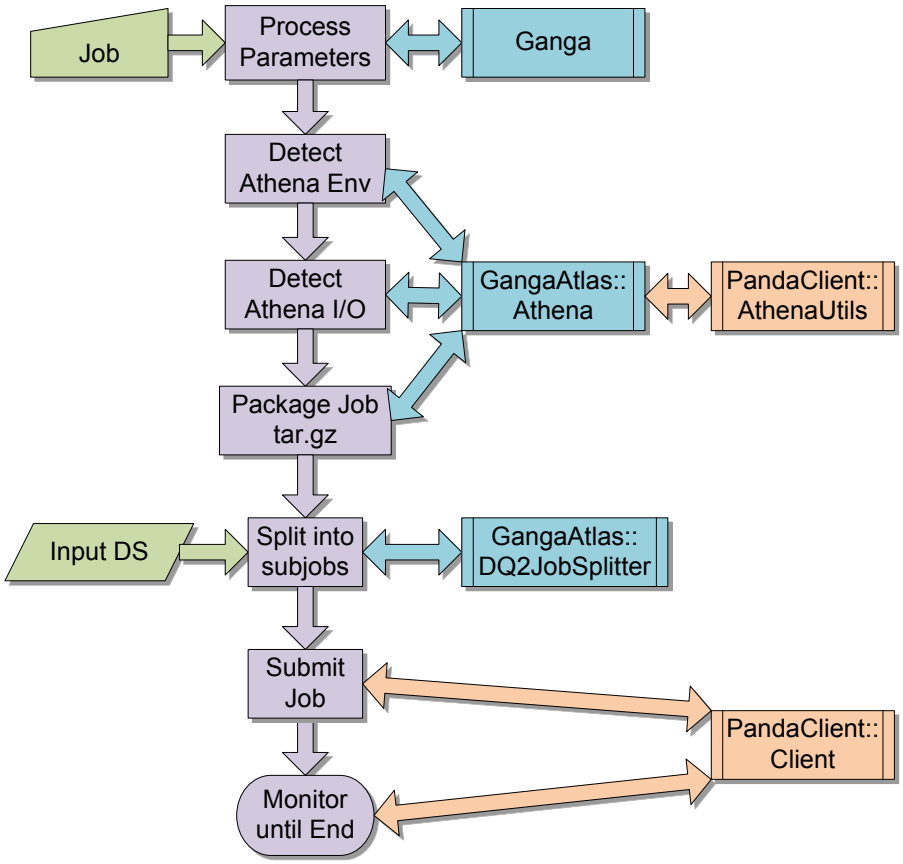

Figure 3. Workflow of the Athena-Panda Runtime Handler, showing the reuse of GangaAtlas and PandaClient components.

The workflow of one such runtime handler (Athena-Panda) is depicted in Figure 3. Generally, the submission of an Athena job to Panda progress through 3 phases. First, the job is packaged into a format suitable for PanDA; second, the large job is split into smaller subjobs to provide a faster turnaround time; and finally, the jobs are sent to PanDA and monitored for their progress. In order to reduce duplication of effort and enable user support, code from the Panda Client and GangaAtlas has been reused to a significant extent. Specifically, the runtime handler reuses the AthenaUtils library in PandaClient in order to detect the Athena environment, detect the the data input and output files, and package the user area for the grid. As of Ganga 5.2, the communication with AthenaUtils happens via the Athena class in GangaAtlas; in this way the job preparation phases are consistent for all Ganga backends.

4. Personal Pilots with GangaPanda

Though the pilot model employed by PanDA provides performance and flexibility benefits, it comes with trade-offs related to local accounting and data security at the grid sites. PanDA operates using a small number of pilot "factories" whose role is to inject generic pilot jobs to the sites; each of the pilot jobs retrieves a real job from the PanDA server and executes it locally. These pilot factories are operated by a small number of people, and the pilot jobs are submitted using their personal grid certificates (which usually have higher level of privileges on the grid sites). Since the pilots are submitted this way, the user jobs do not run under the submitter's certificate, but instead they run with the certificate of the pilot job. This issue can

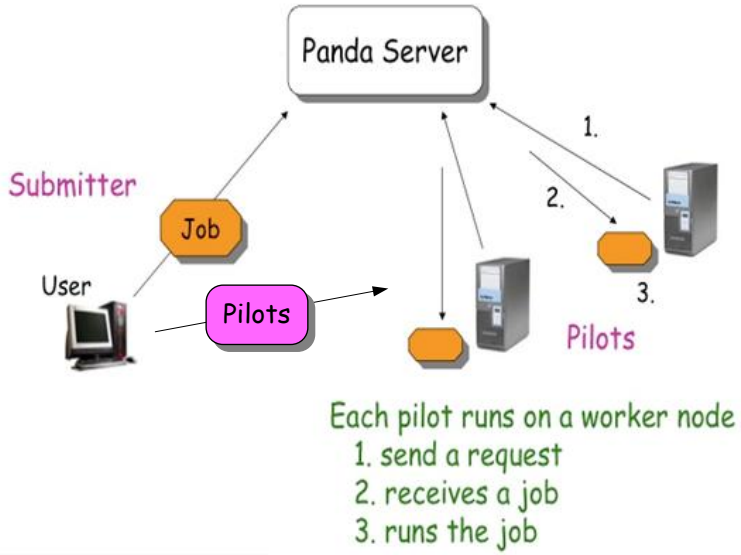

Figure 4. With PanDA, user jobs are first submitted to the PanDA server, and then retrieved by a generic pilot script running on the worker node. To access sites that do not receive pilots from the central pilot factories, GangaPanda allows users to send "personal pilots" directly to the sites via the other backends of Ganga (e.g. LCG, NG, or Batch).

```
myjob=Job()
myjob.application=Athena()
myjob.backend=Panda()
myjob.backend.site='ANALY_FZK'
myjob.submit()
```

```
pilot=Job()
pilot.application=PandaPilot()
pilot.application.queue='ANALY_FZK'
pilot.backend=LCG()
pilot.submit()
```

Figure 5. A personal pilot code example. On the left, a user prepares and submits `myjob` which is an Athena job to run at `ANALY_FZK`. If this site does not receive pilot jobs, the user can directly submit a personal pilot using the code on the right.

cause problems related to local usage accounting and data security, and for that reason some sites have been hesitant to allow pilot jobs to run. GLexec [7], a tool which enables unix user switching based on grid credentials, has been accepted as the solution to this problem; however, delays in its deployment have necessitated alternative temporary solutions. The personal pilot application, known as `PandaPilot` within Ganga, is one such alternative.

Figure 4 shows the workflow of the personal pilots from GangaPanda. When a user submits a job to the PanDA Server, the job waits until a pilot retrieves it from the database. A generic pilot job is then sent from Ganga via the LCG backend to the worker node. The pilot starts running at the site with the user's credentials, the actual job is retrieved from the PanDA server, and then the job proceeds normally.

The example in Figure 5 presents code examples for using Ganga personal pilots. Consider the case where a user wants to process data with a PanDA-based analysis at ANALY_FZK, a site which does not normally receive pilot jobs. A user can easily submit a job to this site (Figure 5 left), but the job will sit at the PanDA server until it expires. In order to retrieve the job, the user can send the `PandaPilot` application to the same site with the LCG backend of Ganga (which invokes gLite middleware [8]). In this simple example, only a single job is submitted;

with a relatively simple script, users could automate the submission of personal pilots as they are needed.

While the Ganga personal pilots allow users to access resources that would be otherwise unavailable to them, they are not a permanent solution. When the a site runs pilot jobs that are limited to a single user, PanDA's ability to effect fair share or otherwise manage resources is diminished.

5. Conclusion and Future Work

The functionality of GangaPanda has been developed and is in production for ATLAS user analysis. With Ganga 5.2, users can run generic executables, Athena, and small scale monte-carlo productions with AthenaMC. The overlap in functionality with Panda Client is quite substantial, though the common library AthenaUtils is presently used by both tools; further developments to this common source code will encapsulate all aspects related to running ATLAS user analysis on the grid.

References

[1] F Brochu et al. Ganga: a tool for computational-task management and easy access to Grid resources. Submitted to Computer Physics Communications. arXiv:0902.2685v1.
[2] 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
[3] PanDA Client: https://twiki.cern.ch/twiki/bin/view/Atlas/DAonPanda
[4] Open Science Grid: http://www.opensciencegrid.org/
[5] EGEE: http://www.eu-egee.org/
[6] NorduGrid: http://www.nordugrid.org/
[7] D Groep, O Koeroo and G Venekamp. gLExec: gluing grid computing to the Unix world. 2008 J. Phys.: Conf. Ser. 119 062032 (11pp) doi: 10.1088/1742-6596/119/6/062032
[8] E Laure et al. Programming the Grid with gLite. Computational Methods in Science and Technology. Computational Methods in Science and Technology, 12(1):3345, 2006.