Grid-based production and operations in the
ATLAS experiment.

F. M. Brochu;*
on behalf of the ATLAS collaboration.

Abstract

We report on the first production use of the ATLAS experiment’s centralised Monte Carlo production system, harnessing the resources of 3 different Grid projects to process 10 million of simulated physics events over the second half of 2004. We will focus on the way we achieved to use these computing resources in a semi-automated way while providing physicists with a common interface to access produced data.

The ATLAS experiment [1] is designed to study the products of high-energy proton-proton collisions at the Large Hadron Collider (LHC), currently under construction at CERN, near Geneva (Switzerland).
It is operated by a worldwide collaboration and is expected to produce over 1 Peta-Byte (or 1000 Tera-Bytes) of data per year. To cope with this vast amount of data to process and the large number of geographically scattered users, a distributed computing model is evolving, relying on the development of a generalised network of computing resources, the Grid.

As no single computing centre can handle this volume of data and users, the collaboration opted for the parallel development of regional computing resources, put together on a global computing network called the Grid or Data Grid.
This global network is not yet in place, although several regional networks exist, forming a local Grid project.

Among the projects aiming to develop such a computing grid, the LHC Computing Grid project [2] (LCG), initiated by the CERN in collaboration with institutes across Europe, is being tested by several HEP experiments including this one.
In UK, the LCG project is supported by GRIDPP [3].

The ATLAS Data Challenges [4] are set to validate the data processing software developed by the collaboration within the computing model mentioned above, and to prepare for the data taking scheduled to start in 2007.

1 The ATLAS Data Challenges.

The second part of ATLAS Data Challenges (DC2) was held during the second half of 2004. The data production took approximately 135,000 SI00 days (Spec Int units, see [5]) and was run on 115 sites located in 22 countries across the world, as shown in Figure 1.

* Cavendish Laboratory, University of Cambridge, Cambridge CB3 0HE, UK.

Keywords: ATLAS Experiment, Data Challenges, Grid, Mass production.
The main entry point is a database (prod DB) to keep the records of all production processes (jobs), their status and their output (backing up the data management system). This database feeds in turn the various Grid with job records which are interpreted and formatted for submission by the Grid Interfaces (super, exe). Each individual job creates an output file which is recorded on the project’s Replica Catalogue (RLS).

The central data management service (DMS) is a layer on top of the different Replica Catalogues, allowing data location and transfers across and between the Grid projects.

3 The Grid Interfaces.

The Grid Interface is the actual processing layer of the production system. It requires an operator with a valid grid certificate registered in one of the three grid projects and the associated Grid User Interface (or a machine with the relevant Grid project client middle-ware). The interfaces are semi-automatic by design: the operator is meant to perform only maintenance operations (like the grid proxy renewal and routine checks). The production is run on the Grid project using the operator certificate.

The Grid interface is made of two components: a common Supervisor and a dedicated Executor.

3.1 The supervisor layer.

Windmill [8] is an interface which deals directly with the production database and feeds automatically its attached executor with predigested production requests.

It manages the timing of all orders, or events, sent to either Prod DB (queries and updates) or the executor (job cre-
ations and status updates). It uses Jabber [9] as asynchronous communication protocol with the different executors.

3.2 The actual Grid interfaces: the executors.

All executors take the same common job data (coded in XML format) from the supervisor and convert it into a job which is submitted on their respective Grid. They also have the responsibility of managing and monitoring all the jobs being processed on their Grid.

Given the structural differences between the supported Grid projects, a given executor works for only one Grid project. But all of them use the common Supervisor interface.

3.2.1 LCG executor.

This application is written in Python, using the LCG User Interface libraries for job submission and monitoring.

It is highly automated. The operator just need to start/stop the duo supervisor/executor and watch the production going. Job submission, matchmaking, monitoring and output retrieval are fully automatic, but can be tailored by the operator (who can exclude/target processing sites for instance).

The price to pay for this high level of automation is a low efficiency (~40%), mainly due to the inability to automatically spot and reject “job sinks”, sites that are accepting jobs while being unable to process them.

3.2.2 NorduGrid executor.

The NorduGrid executor is a C++ application interfacing with NorduGrid User Interface API and GLOBUS RLS API (for data management). On top of job management, this executor is handling the registration of output meta-data into the NorduGrid RLS, as a validation routine.

The NorduGrid executor is the most robust and scalable interface created for this project. Like the LCG executor, the operator is just needed for routine maintenance. For the whole DC2 operation, no more than two operators at a time were needed to run the production on NorduGrid, with an efficiency twice larger than LCG. This high efficiency comes mainly from the overall stability of the Grid project itself.

3.2.3 Grid3 executor.

This Python application sits on top of a bunch of dedicated Grid applications which aim to reproduce the functionalities of a User Interface. This application prepares the job submission (job description file) and the Grid applications handle the submission, monitoring and retrieval of the jobs.

The actual job submission and selection of the processing site are left to the operator. Efficiency numbers at the level of NorduGrid were achieved, thanks to the presence of operators who could timely spot “job sinks” (see above) and avoid them.

4 Uniform data access.

The Data management service is a set of Python modules called Don Quixote (DQ) [10]. It is made of three servers (one per Grid project) for actual data management and services (data replication and location) and a Client side to handle users requests (as described in Figure 3).
server were added to provide automatic mass replication of data, mainly used to replicate the produced data from disk Storage Elements to tape. Beyond data production, ATLAS users can retrieve any data file in a transparent way by using the client side of Don Quixote. User grid certificate is required for data downloads.

5 Summary.

We have described a heavy weight, centralised application which allows a small group of people to run a mass data production over the resources of three Grid projects and access the data produced in a transparent way. Although this does not provide a complete solution for cross Grid operation, this can be considered as a first step in this direction.

References

[10] https://uimon.cern.ch/twiki/bin/view/Atlas/DonQuijote

Fig 3: Don Quixote Layout.
The client side sends the requests to the Aggregator, which route them to the appropriate Grid server. The later translates and executes the orders in the framework of the related Grid project. Cross-grid replication is performed through GridFTP. Later on, a dedicated database and client