ATLAS PRODUCTION SYSTEM IN ATLAS DATA CHALLENGE 2

L. Goossens, CERN, Geneva, Switzerland
K. De, UTA, Arlington, USA

Abstract

This paper describes the design, the implementation and experience with the ATLAS production system in ATLAS data challenge 2 (DC2). The system, actually a suite of collaborating components, was designed based upon the experience gained during DC1 (summer 2002-spring 2003). The major design objectives were set to be: minimal human involvement, maximal robustness, and last but not least interoperability with the three grid flavours in use within ATLAS and legacy systems. A non-negligible change of spirit was the switched roles of grid and non-grid support. The DC1 systems were designed to be able to run on legacy resources and also run on the grid. The DC2 production system was designed to run on the three grid flavours used in ATLAS, and as a backup solution on legacy resources.

INTRODUCTION

Like the other LHC (Large Hadron Collider) experiments, ATLAS is undertaking a series of data challenges to validate its computing model, its data model and its software. The second in this series, DC2, started July 2004 and is planned to finish in early 2005.

DC2 is organised in three phases. In phase one, more than 10 million events belonging to about 30 different physics channels will be fully simulated. These events will be digitized (partly at zero and partly at design luminosity), mixed, and converted into a byte stream format similar to what is expected to come out of the on-line system. This phase started July 2004 and is expected to finish by November 2004.

The second phase, called the tier-0 exercise, will test the part of the computing model concerned with the prompt reconstruction and the distribution of the data to tier-1 and tier-2 centres. The third phase will test the part of the computing model addressing distributed analysis.

In the previous DC1 ATLAS deployed two production systems. The first system, AtCom [1], supported interactive production through a convenient graphical user interface. The design of AtCom was based on a generic core and a number of plug-ins interfacing to various legacy batch systems and/or grids. Although very successful, it was clear that AtCom, by design, was not suited to manage 10,000's of jobs. The second system, GRAT [2] was deployed in the US and was a suite of scripts used to manage jobs on a grid consisting of seven sites. GRAT used the Globus toolkit for grid middleware. In spite of the success of GRAT in DC1, it was too simple and too specialized for use in DC2.

Based on the experience with these two systems ATLAS decided to develop a new automatic production system to be deployed on the time scale of DC2.

CONCEPTUAL MODEL

Before starting with the presentation of the actual design of the production system, it is worthwhile to briefly present the overall conceptual model and terminology in use within the ATLAS production context.

The production system distinguishes between two levels of abstraction (see figure1). On the higher level, input datasets are transformed into output datasets by applying a task transformation. The process of doing this is called a task.

Datasets are usually quite big and realized with a large number of logical files. On the lower abstraction level, input logical files are transformed into output logical files by applying a job transformation. This process is called a job.

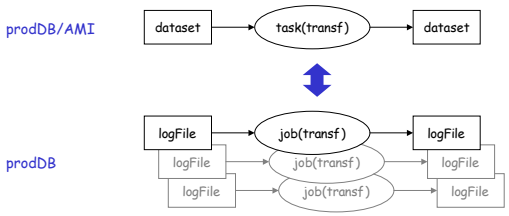

Figure 1: Conceptual model.

ARCHITECTURE

The architecture of the production system was designed with the following goals in mind.

1. The system should be as simple and as flexible as possible.
2. The system must target automatic (minimal human intervention) production on each of the three grids in use within ATLAS, i.e. LCG, NorduGrid and GRID3. Legacy batch systems must be supported as well as a backup solution.
3. The system should use GRID middleware services as much as possible.

The resulting design is shown in figure 2. It is based on four components: a central production database (prodDB), a data management system (dms), a supervisor component and an executor component. In the following subsections we will address each of these components separately.

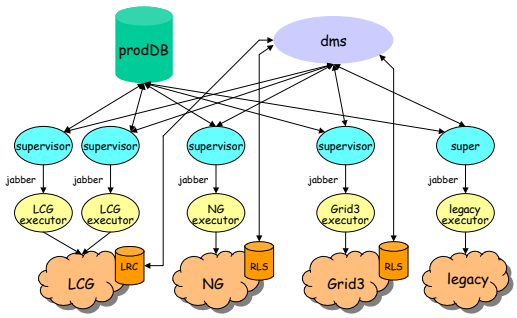

Figure 2: Architecture.

By adopting a component oriented design and additionally allowing the components to run as agents on different servers, communicating asynchronously with each other, a lot of flexibility, both with respect to the logical composition of the system and to its physical realization was made possible.

Note that the design relies heavily on grid services like the Globus replica location service (RLS), the LCG information system, the LCG job submission broker, etc. On the positive side this meant we did not have to invest effort in producing our own equivalents of these systems. On the negative side, it was not clear whether these services would actually work and relying on them introduced a considerable risk (and indeed we discovered the hard way that many of them did not work).

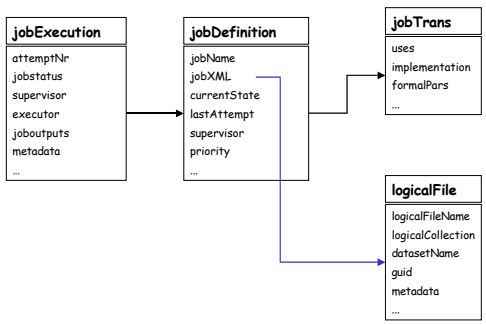

Figure 3: Tables in production database.

Production database

There is only a single logical production database. The physical realization of the database may be distributed and/or replicated, but to the other components in the design it will look like a single entity.
The database holds tables with records for

- job transformations
- job definitions
- job executions
- logical files

The relations between these tables are shown in figure 3.

A job transformation record describes a particular combination of executable and release. The description includes the signature of the transformation listing each formal parameter together with its type (restricting the possible values) and its meta-type (indicating how the values should be passed to the executable).

Each job definition record points to its associated job transformation. Other fields allow one to keep track of the current attempt at executing this job (lastAttempt), which supervisor component is handling this job (supervisor), what is the relative priority of this job (priority), etc. The bulk of the job definition is however stored as an XML tree in the field jobXML. It lists the actual values to assign to the formal parameters of the transformation and additional information about logical input files and logical output files.

For each job definition there can be zero, one or more job execution records corresponding with each attempt at executing the job. Each attempt has a unique number which is appended to the names (both logical and physical) of all files produced, ensuring interference-free operation even in the case of lost and/or zombie jobs. The execution record also records information like start and end time of the job, resources consumed, where the outputs were stored, etc.

In the last table, logicalFile, the production system stores all meta-data about logical files. Most of the information is redundant with respect to the information stored in the respective meta-data catalogues of the grids (size, guid, md5sum, logicalCollection), but at the time the production system was developed these meta-data catalogues did not support schema evolution and ATLAS did not a priori know what meta-data was needed. Consequently, it was decided to temporarily deploy our own catalogue in addition to filling and using the existing ones.

The DC2 production database was an Oracle database hosted at CERN.

Supervisor

The next component, called supervisor, takes free jobs from the production database and hands them out to one of the executors it is connected with. The information about jobs is exchanged using XML, usually wrapped in

XMPP (using the Jabber protocol) or wrapped in SOAP using web services. As its name suggests, the supervisor will follow up on the job, asking at regular intervals about the job status until the job is declared 'done' by the executor. At that point, the supervisor will, for successful jobs, verify the existence of all expected outputs, and, if all is as expected, will rename them to their final logical name (by dropping the attempt number from their temporary logical name). Additionally, the files will be added to the logicalFile table together with any meta-data produced by the job and returned by the executor. In the case of a failed job the supervisor will simply release the job in the production database, so that it can be picked up again if the maximum number of attempts is not yet reached.

The supervisor does not perform any brokering. The handing out of jobs is based on a simple "how-many-do-you-want" protocol. The supervisor asks the executor how many jobs it wants (possibly qualified with resource requirements) and the executor replies with a number (possibly qualified with, not necessarily the same, characteristics). The supervisor may then send a number of jobs to the executor, which in turn may choose to process or refuse them. The non-binding nature of the protocol allows both very sophisticated and very simple implementations to co-exist on both the executor and supervisor side.

For efficiency reasons an implementation of the supervisor can keep state but the design does not require this. Having a stateless component obviously makes it more resilient against crashes.

The supervisor implementation for ATLAS is called Windmill [3]. Each Windmill instance connects with a specific executor, and manages all jobs processed by this executor. Since Windmill is stateless, it can be robustly reconnected to the same executor without loss of job status information

Executor

The task of the executor is to interface the supervisor to the different grid or legacy systems. It translates the grid/legacy neutral job definition into the grid/legacy specific language (xrsl, jdl, wrapper scripts, …), possibly adding some pre and post processing steps like staging in/out of files. The executor implements a grid/legacy neutral interface with the usual methods: submit, getStatus, kill, etc. Again the design does not require the executor to keep state.

Four executors were developed and deployed within the context of DC2

- Dulcinea [4] for the NorduGrid
- Capone [5] for Grid3
- Lexor [6] for the LCG
- a set of similar executors for legacy batch systems like LSF, PBS, BQS [7]

Data Management System

The data management system fulfils two functions: global cataloguing of files and global movement of files. In ATLAS we have opted to realize the global catalogue function by building on the existing catalogues of the three grid flavours (Globus RLS in the case of NorduGrid and Grid3, LRC (local replica catalogue) in the case of the LCG). The data management system acts as a thin layer channelling catalogue requests to the respective grid catalogues and collecting/aggregating the answers. At the same time it presents the users with a uniform interface on top of the grid native data management tools, both for the catalogue functions and the data movement functions.

The implementation of the data management system used in ATLAS is called Don Quijote [8]

EXPERIENCE

Between the start of DC2 in July 2004 and the end of September 2004, the automatic production system has submitted about 235K jobs belonging to 158K job definitions, producing about 250K logical files. These jobs were approximately evenly distributed over the three grid flavours. The definitions belonged to 157 different tasks, exercising 22 different transformations. Overall, they consumed ~1.5 million SI2K months of cpu (~5000 cpu months on average present day cpu) and produced more than 30TB of physics data.

By design, the production system was highly dependant on the services of the grids it interfaces to. This was known to be a risky dependency from the beginning and indeed we suffered a lot because of it. The Globus RLS deployed by both NorduGrid and GRID3 turned out to be very unstable and became reasonably reliable only after a series of bug fixes. We had a similar experience with several of the LCG services, e.g. the resource broker and the information system. Because the LCG is by design the most complex system of the three grids requiring many services to work at the same time to do anything, it is not surprising that this resulted in the highest failure rate of the three grids.

But it was not only the grid software that needed many bug fixes. The data challenge started before the development, let alone the testing, of the ATLAS production system was finished. As a result, various bugs had to be corrected, and new features introduced, during the data challenge.

More detailed experience reports can be found in [4], [5] and [6].

CONCLUSION

For the second in its series of data challenges, ATLAS relied/relies completely on a federation of grids: NorduGrid, GRID3 and LCG. The ATLAS production system was designed for automatic production on this federation of grids. By design the system relied heavily on the services offered by the grid systems. Stress-testing these services in the context of a major production exercise was a new experience and many lessons were learned.

It was possible but a lot of manpower was needed to compensate for missing and/or buggy software.

ACKNOWLEDGEMENTS

The authors would like to thank the many people involved with running the ATLAS data challenge 2.

REFERENCES

[1] V. Berten, L. Goossens, C.L. Tan, "ATLAS Commander: an ATLAS Production Tool", proceedings of CHEP03, 2003.

[2] R. Sturrock et al. – ATLAS Collaboration, "A Step Towards A Computing Grid For The LHC Experiments : ATLAS Data Challenge 1", CERN-PH-EP-2004-028, CERN, Geneva, 30 Apr 2004.

[3] http://heppc12.uta.edu/windmill/

[4] R. Sturrock et al., "Performance of the NorduGrid ARC and the Dulcinea Executor in ATLAS Data Challenge 2", proceedings of CHEP04, 2004.

[5] M. Mambelli et al., "ATLAS Dara Challenge Production on Grid3", proceedings of CHEP04, 2004.

[6] D. Rebatto, "The LCG-2 Executor for the ATLAS DC2 Production System", proceedings of CHEP04, 2004.

[7] J. A. Kennedy, "The role of legacy services within ATLAS DC2", proceedings of CHEP04, 2004.

[8] M. Branco, "Don Quijote – Data Management for the ATLAS Automatic Production System", proceedings of CHEP04, 2004.