The role of legacy services within ATLAS DC2

Dr. J. A. Kennedy, LMU, Munich, Germany

Abstract

This paper presents an overview of the legacy interface provided for the ATLAS DC2 production system. The term legacy refers to an interface to a batch system, such as PBS, for the ATLAS production system.
In a world which is becoming increasingly grid orientated this project allows us to evaluate the role of non-grid solutions in dedicated production environments. Experiences, both good and bad, gained during DC2 are presented and the future of such systems is discussed.

INTRODUCTION

The ATLAS Experiment

The ATLAS experiment at CERN will begin data taking in 2007 when the LHC is commissioned. From then on data is expected to be recorded at a rate of 2 PetaBytes per year with an additional 1 PetaByte of simulated data produced per year.

This huge data volume means that it is impossible to perform all the processing and analysis at CERN. ATLAS aims to utilise distributed computing resources in collaborating countries from all around the world.

To aid with the development of this distributed computing infrastructure a series of data challenges were started in 2002. The data challenges allow us to evaluate the ATLAS computing model, the full software suite, the data model and also to ensure the correctness of technical choices made for ATLAS computing.

Overview of Data Challenge 2

The second data challenge, DC2, began in summer 2004. DC2 focuses on the use of an automated production system and GRID software. The goals of DC2 can be summarised as follows,

- Produce 10^7 Events
- Use Geant4, POOL, GRID
- Pileup/Digitization in ATHENA(The ATLAS software framework)
- Complete Event Data Model and Detector Description
- Full ATLAS Simulation + Combined TestBeam
- Use/Test GRID middleware
- Large Scale Physics Analysis
- Computing Model Studies

The task of producing simulated data for DC2 was split into several subtasks such as event generation, simulation and reconstruction.

Different paths can be taken through the production depending on which final simulated data is required. This modularity is very useful and powerful within a production system.

The data challenge was split into three distinct phases

- Part 1: Production of Simulated Data
  - Geant4, digitization, pileup, POOL
  - minimal reconstruction to validate
- Part 2: Test of Tier-0
  - Full reconstruction
  - ESD+AOD distributed in real time to Tier-1s for analysis
- Part 3: Distributed Analysis
  - Access to event and non-event data from any facility, organised and chaotic approaches

This splitting of the data challenge allows a natural task flow through data production to the distributed analysis challenge.

ATLAS PRODUCTION SYSTEM

The production system for DC2 is designed to provide a common framework in which any grid flavour or legacy system may be integrated[1, 2, 3]. The production system is formed from several individual elements which when plugged together provide the required functionality for the submission, tracking, recovery and validation of jobs. The individual elements of the production system may be summarised as follows.

- Common database for production jobs
- Common Supervisor run by all facilities/managers
- Executors developed by middleware experts
- Data Management system to allow intergrid data transfer and file cataloging

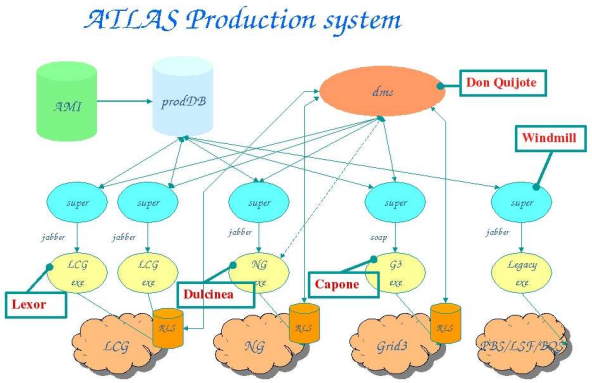

Figure 1: Overview of the ATLAS production system.

Figure 1 shows an overview of the ATLAS production system. The core of the ATLAS production system is formed from the coupling of a supervisor with an executor. The supervisor provides an interface to the job definition data and metadata associated with the jobs while the executor provides an interface to the computing resources. The communication between the two elements is performed using either Jabber or SOAP, this allows some flexibility in the design and implementation of the system. The use of XML messaging to communicate between the supervisor and executor provides a means of allowing the individual elements of the production system to be designed without being strictly tied to one technology. The supervisor interfaces to a central production database and retrieves job specific information. The supervisor-executor system then allows these jobs to be passed to one of the grid flavours or standard batch system. Through it's interface to the executor the supervisor may continuously monitor the state of the submitted jobs and finally retrieve detailed information about the job once the job has finished.

Although the system currently uses a one-to-one supervisor-executor mapping, one-to-many and many-to-one mappings may also be deployed.

THE LEGACY EXECUTOR

Despite DC2's commitment to deploying grid solutions whenever possible, several considerations lead to the development of a Legacy executor. The Legacy executor may be used as a backup solution should unforeseen problems arise with the development of grid executors, provide testing facility for use when developing the production system and to provide access to computing resources at sites which have not yet deployed any grid middleware.

Overall aims

The Legacy executor is designed primarily to be simplistic and provide an adaptable system which may be ported to any batch flavour. In spite of this aim for simplicity it is inevitable that different environments and different batch systems deployed at sites will require some re-configuration of the executor. The development aims to keep this reconfiguring to a minimum. The Legacy executor aims to fit seamlessly into the production system providing the standard interface to the supervisor and to all intents and purposes be indistinguishable from a grid implemented executor.

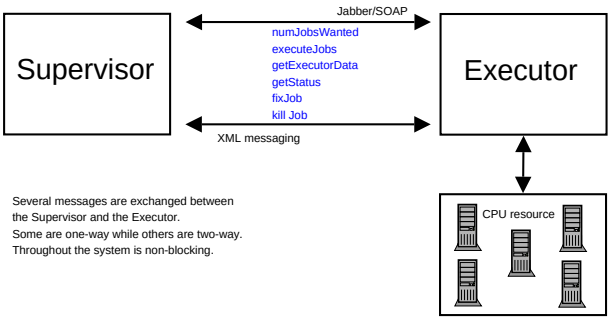

Figure 2: The Supervisor-Executor system, showing their communication via XML messages and the executors coupling to a set of CPU resources(a batch system in the case of the Legacy executor).

Implementation

The Legacy executor was developed using python thus allowing for rapid development and testing. The supervisor project development was also undertaken using python and provides a simple executor template which was utilized during the development of the Legacy executor. In addition several supervisor tools for the interpretation, processing and creation of XML messages were also used by the Legacy executor. The executor allows some site specific configurations values to be set such as the batch queues to interface to and the maximum number of executor jobs to allow into these queues.

The Legacy executor implements the standard six methods required for communication with the supervisor.

- numJobsWanted
- executeJobs
- getExecutorData
- getStatus
- killJob
- fixJob

Figure 2 shows a schematic of the coupling of a supervisor to an executor and their communication. The XML messaging enables the communication between the supervisor and executor to be performed in a non blocking manner.

The standard cycle of job reservation-submission-monitoring-validation is performed using 4 of these methods, numJobsWanted, executeJobs, getExectorData and getStatus. Each of these four messages is continuously polled from a supervisor to an executor with all the jobs associated with this state.

Figure 3 shows a schematic of the flow through each of the 6 methods within the Legacy executor. Several of the methods share common requirements and thus common core elements were developed within the Legacy executor, helping reduce redundancy and also aiding with portability to other batch systems.

numJobsWanted In numJobsWanted the supervisor asks how many jobs of a particular specification the executor requires. The job specification can provide information about requirements such as CPU time, required memory etc. The executor queries the batch system and the configuration data and returns the number of jobs required.

executeJobs Once jobs have been requested the supervisor sends executeJobs and the executor then takes the data associated with each job creates a jobInfoObject and archives it. A wrapper script is then created for the job and this is then submitted to the batch system with the local jobId being recorded for later reference. The local jobId is then mapped to the global jobId provided by the supervisor.

getExecutorData Once jobs have been submitted, via executeJobs, the supervisor asks getExecutorData, the executor queries the status of the specified jobs and returns this job status to the supervisor.

getStatus Once Jobs have started to run the supervisor will ask getStatus and the executor will again query the batch system for the job status. However with getStatus an extended level of information is returned to the supervisor, information such as the names,sizes,md5sums of output files as well as diagnostic information about the job exit code,job time, compute node etc, is returned to the supervisor. The availability of the information is dependent on the status of the job itself, if the job is running this is reported and null information is reported for fields which are unavailable otherwise more detailed information is found and returned to the supervisor.

The killjob and fixJob methods are not currently used within the standard job cycle. However a minimum implementation of these has been provided within the legacy executor.

KillJob The killJob call is simple in its implementation with a map from the global jobId to the local jobId required and then an interface to the batch system to kill this job is made.

FixJob The fixJobs method was originally envisaged to allow several different functionalities, job cleaning, job resubmission etc. The legacy executor has implemented only a simple job re-submission method. The global jobId is used to retrieve the archived jobInfoObject and this is then used to re-submit the job via the executeJob method, the number of attempts associated with the job is incremented. Job re-submission is only allowed by the executor if the number of submission attempts is below a used defined maximum.

In addition to ensuring that the Legacy executor provided the functionality to interface with the supervisor and run jobs on a local batch system it was considered a design goal to ensure that it also provide a adequate level of persistency and fault tolerance.

The executor is made persistent by storing the global to local job mapping and allowing queries access to this store, thus when the executor fails a persistent store of job information remains intact and upon re-launching the executor the correct information about the assigned jobs can be obtained. This persistency is very important in a distributed system to ensure that failures in the system don't cause the loss of jobs at remote sites.

Deployment and testing

The Legacy executor was primarily developed with a PBS batch system at the FZK center in Germany. At various points during the development process the executor was deployed at alternate sites and adapted to their systems. Sites used include RAL(PBS) and CERN(LSF). Once a working framework for the legacy executor was available a parallel development was undertaken using the CCIN2P3(BQS) system. The deployment at different sites allowed the executor to remain focused on it's goal of providing a generic system with minimum required alterations for different sites.

The PBS based executor for the FZK site was tested extensively with both event generation and simulation jobs. In the final testing phase the executor was run with a continuous float of 200 jobs, with 400 jobs having a typical duration of around 24hrs, being processed during this test. Although this number is somewhat smaller than the numbers available when using grid based executors it is nevertheless convincing for a single batch system and it is envisaged that the executor would be capable of scaling to even large batch system capacities.

The persistency and resistance to executor/supervisor failure was tested by killing either the executor or supervisor once jobs had been submitted to the batch system. The majority of jobs could be successfully retrieved although in some cases the jobs were lost. More investigation is required into the mechanism by which jobs are lost and to ensure that this possible cause of failure is corrected.

EXPERIENCE

Several positives came from the development of the Legacy executor as well as a some indications about the limitations of such systems in production environments. One positive is the ability to aid with the testing and feedback cycle. The development of a simple Legacy interface allows for fast feedback to the developers of the production system. This is somewhat limited however since the legacy executor cannot identify all possible problem areas since several grid specific areas are not covered. The interaction

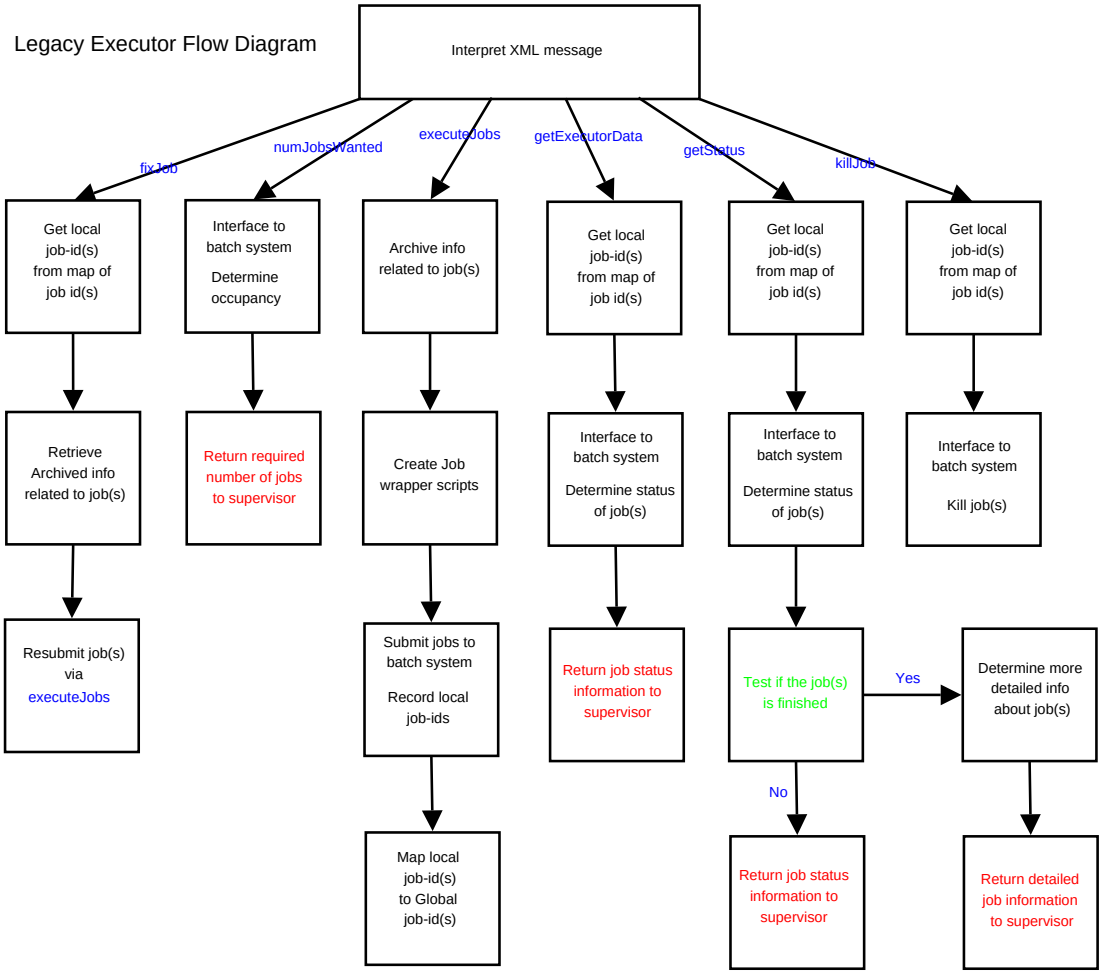

Figure 3: Flow Diagram for the Legacy Executor. The XML message interpretation is performed using tools from the supervisor, once this is performed the task associated with the message is executed.

of the supervisor and executor as well as the functionality of the supervisor are areas which legacy systems can help test. On the side of limitations probably the most striking is the lack of tools for file cataloguing and replica management. As the project progressed these grid tools were used more and more and data management tools were directly used within the supervisor. This functionality is lacking within the Legacy executor and was a problem area.

CONCLUSIONS

The development of the Legacy executor has shown that simple interfaces to batch systems are still viable for large scale production systems but their usage comes at a cost. The lack of file replication and management tools means that they lack some functionality which many view as essential for a production system. This leads us to a choice, we can have a simple batch interface and sacrifice some functionality or we can start to add this functionality and design something which isn't so simple anymore. Doing away with simplicity may lead to a system which suffers much the same problems as the grids themselves and moves away from being a "backup" solution. One area in which a legacy interface may prove to be very valuable is the initial production and testing phase. A simple solution can be produced relatively quickly and can greatly benefit the early development of a project.

ACKNOWLEDGEMENTS

I would like to thank the members of the ATLAS production team, in particular Kaushik De and Luc Goossens, for their help and comments during the production of the legacy executor. I also would like to acknowledge the work of Karim Bernardet who undertook the parallel development of the BQS based executor.

REFERENCES

[1] L. Goossens, ATLAS Production System in ATLAS Data Challenge 2, Luc Goossens, These Proceedings

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

[3] M. Branco, Don Quijote - Data Management for the ATLAS Production System, These Proceedings