ATLAS Distributed Analysis (ADA)

D. L. Adams, K. Harrison, **R.W.L. Jones**, A. Soroko, C. Tan
Brookhaven Laboratory/Cambridge University/Lancaster University/
Oxford University/University of Birmingham

Abstract

The ATLAS distributed analysis system is described. The major components of this system are web services to manage processing, catalogue services to record data and its provenance, clients to interact with these services and an abstract job definition language used to format messages between clients and services.

1. Introduction

ATLAS is the largest of the experiments planned for the Large Hadron Collider (LHC) being constructed at CERN. The experiment will record approximately 10^9 events per year. Each event represents a readout of the detector for a triggered collision and the average raw event size will be 1 to 2 MB. Each event will be reconstructed at least once yielding another 0.5 to 1.0 MB of data for each event. The total volume of data produced will be 5 to 10 PB per year.

At present there are 1800 physicists spread over more that 150 universities and laboratories in 34 countries participating in the experiment. It will not be possible for these scientists to freely access all of the data. Instead the data processing is split into two phases: controlled production, where the raw data is reconstructed to produce derived data, and analysis, where small groups or individuals access the derived data.

The computing resources and the data will be distributed over dozens of sites and a grid-based computing model is being developed to provide access to the data and computing resources.

The ATLAS distributed analysis system (ADA[1]) faces the challenge of supporting distributed users, data and processing. It must support all analysis activities that can be sensibly distributed, including Monte Carlo production, data reconstruction and the extraction of summary data from all types of event samples. It is required to work across three current major grid deployments: LCG[2], Grid3[3] and NorduGrid[4] and the future systems into which these evolve. An important example of the latter is that being developed by EGEE[5].

2. Model

The model for ADA is illustrated by the block diagram in figure 1. On the bottom are generic middleware services that can be shared with other experiments, other fields of science and other academic and business applications. These include services such as replica catalogues, metadata catalogues, computing and storage elements and workload management systems.

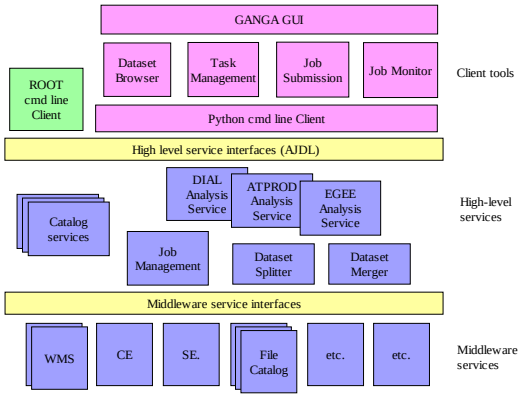

Figure 1: The ADA layered architecture

The middle layer contains high-level services whose implementation depends on the domain (ADA in our case). An important goal of ADA is to generalize requirements and express them in terms that are or can be addressed by the middleware providers. The high-level services present a view natural to the users and delegate most of the work to the middleware services.

This layer includes catalogue services where users can go to find data, transformations used to create data and associated metadata. Another important feature is the analysis service which carries out the processing used to create data or analyze existing data. The figure also shows a job management service that enables users to

interact with processing jobs and shows splitters and mergers used to distribute processing and merge results.

The top layer of the system holds the clients which present a command line or graphical interface to the user and relay requests to the high-level services. Communication between the clients and services is expressed in terms of AJDL (Abstract Job Definition Language).

A prototype system has been created based on the GANGA[6] user interface and the ATLAS production[7] and DIAL[8] job management systems. Work is also underway to incorporate middleware from EGEE in the context of the LCG ARDA[9] project. Details of some of the components are discussed in the following sections.

3. AJDL

AJDL defines the terms in which users interact with ADA. It provides classes which can be used to construct clients and services and XML representations of the data in these classes so that objects may be catalogued and exchanged between clients and services.

AJDL components have base types and subtypes which extend the former in data, interface and/or implementation. An important goal is for the base types to be sufficiently rich that analysis services may be constructed from these and so can be used with any or most subtypes.

The central component of AJDL is the dataset. A dataset has an identity, has content describing the kind of data it holds, may have a location for its data (e.g. a collection of logical files) and may be compound (i.e. hold a list of sub-datasets). Users may locate an existing dataset in the dataset catalogues and, if sufficiently small, copy the dataset and its data to a local disk for direct inspection or processing.

When the dataset is large, the user may select or define a transformation to act on the dataset to produce another dataset of more manageable size, e.g. by selecting events or summarizing the data in events. The AJDL transformation consists of an application which specifies the executable and environment for processing and a task which carries data used to configure the application at run time, e.g. input parameters or an algorithm to compile and load.

The application provides two scripts: the build script which readies the task for processing and the run script which acts on an input dataset to produce a result dataset. At present the application is specified with a name and version and each processing site is assumed to have installed equivalent scripts at locations mapped by name and version. In the future scalability and consistency will be improved by embedding the scripts in the application and providing standard means to locate software at each site.

The present task carries a collection of named text files. This may be extended by adding named parameters and allowing the option of referencing the files with logical file names.

A job is defined by identifying an input dataset, a transformation (application plus task) and optionally a set of user preferences. The latter can include information such as the user VO and role, suggested location for output data and desired response time. This job definition is presented to an analysis service which executes the job and responds to status queries with a job description that includes the original definition plus status information such as the state (initializing, running, done, failed, …), start and stop times, list of sub-jobs, etc. The job may also hold a reference to the result dataset when available. This may be a partial result if the job has not completed.

DIAL provides a C++ implementation of AJDL base types and many ATLAS-specific sub-types. It also provides DTD files and means to translate between XML descriptions and C++ objects. GANGA provides a python wrapping of these classes and is implementing some as pure python classes.

4. Analysis services

As described in the preceding section, an analysis service provides means to apply a transformation to an input dataset to produce a result dataset. ADA is developing multiple implementations of the analysis service to provide different capabilities (interactive vs. batch, local vs. grid) and to test different workload management systems (batch queues, ATLAS production, EGEE). Here we describe a few of the existing approaches.

The DIAL project has the goal of studying interactive analysis and has delivered a service which can fork jobs, can submit them to a local LSF or Condor queue or can use Condor COD (computing on demand) to avoid the usual batch latencies. With appropriate choice of queue, this service is appropriate for batch-like processing of long-running jobs or interactive (i.e. low latency) processing of short jobs. The service provides the capability to split the input dataset,

process the pieces in parallel and then merge the results.

ATLAS has developed a production system to carry out the processing of data in the recent data challenge. This system features a database of job requests. Production services running on each of the three ATLAS grids (NorduGrid, Grid3 and LCG) fetch job requests from the database and update the database when the job is completed. An analysis service is being developed that carries out processing by populating this database and polling until the jobs complete. This service is an extension of the DIAL service described in the previous section. It will likely be limited to those transformations that are part of standard production. These are expected to account for a large fraction of the user processing requests.

The production services have two components: a grid-specific executor that carries out processing on the grid of interest and a common supervisor which provides communication between executor and database. There is a separate project to develop an analysis service that connects directly to an executor bypassing the production database. This will likely be developed in python and make minimal use of DIAL. It will not provide job splitting and merging but can be called from a DIAL service when this functionality is desired.

ARDA is developing an analysis service based on the EGEE middleware and in particular on its workload management system. This will start from the DIAL implementation and allow processing at any EGEE-compliant site.

5. Catalogue services

AJDL objects are stored in repositories that provide access to their XML descriptions indexed by ID. There are also metadata catalogues that allow users to make queries to identify objects (e.g. datasets) of interest.

The ATLAS Metadata Interface system, AMI[10], provides the database infrastructure to maintain these catalogues and provides a generic web service interface for making queries. It also provides web pages for guiding users though these queries.

6. Clients

The high-level services have standard interfaces so that clients and services may be developed independently. The end user is free to mix and match clients and services. Clients provide users with means to interact with services and AJDL objects.

DIAL is implemented in C++ and so provides a API in that language. Rootcint is used to construct dictionaries and make this API available at the ROOT command line. It is possible to select an application, task and dataset, submit a job to a local scheduler or any analysis service and monitor progress of the job. ROOT is an especially useful environment when the result dataset includes ROOT objects such as trees, ntuples or histograms.

DIAL also provides a limited command line API for job submission and simple monitoring.

The primary goal of GANGA is to provide a user-friendly environment for job submission and monitoring. GANGA is based on python to enable rapid development, provide platform independence and allow easy extensibility. GANGA provides local job management, including splitting and merging, with primary emphasis on the Gaudi and Athena applications developed by LHCb and ATLAS. It features a graphical interface that leads users through the steps required to create a job and then provides an intuitive panel for monitoring progress of the job.

ATLAS has opted to use the previously described model to move much of the job management to the analysis service but still looks to GANGA to provide the graphical interface for job submission and monitoring. Work is underway to reconcile the GANGA and AJDL job models to make this possible. An important byproduct is the development of a python binding for AJDL. This binding can be used in other applications such as the ATLAS executor analysis service described above.

7. Conclusions

ATLAS is developing a distributed analysis system that is very modular and thus capable of adapting to the evolving requirements of its scientists. Major components are the analysis and catalogue services and the clients presenting both command line and graphical interfaces to users.

8. Acknowledgments

The authors gratefully acknowledge the support of the PPDG and GridPP projects.

9. References

1 http://www.usatlas.bnl.gov/ADA
2 http://lcg.web.cern.ch/LCG/
3 http://www.ivdgl.org/grid2003/
4 http://www.nordugrid.org/
5 http://egee-intranet.web.cern.ch/egee-intranet
6 http://ganga.web.cern.ch/
7 http://atlas.web.cern.ch/Atlas
8 http://www.usatlas.bnl.gov/~dladams/dial
9 http://egee-intranet.web.cern.ch/egee-intranet
10 http://lcg.web.cern.ch/LCG/peb/arda
11 http://isnpx1158.in2p3.fr:8180/AMI