DIAL: DISTRIBUTED INTERACTIVE ANALYSIS OF LARGE DATASETS

D. L. Adams, T. Maeno, Brookhaven National Laboratory, Upton, NY 11973, USA
K. Harrison, University of Cambridge, Cambridge, UK
G. Rybkine, Royal Holloway College, University of London, Egham, UK
D. Liko, CERN, Geneva, Switzerland

Abstract

This document describes the status of the DIAL[1] (Distributed Interactive Analysis of Large datasets) project. We describe the software that is presently available, the deployment of the system for ATLAS[2], and results obtained using that deployment.

INTRODUCTION

The goals of DIAL remain as described in the first DIAL paper presented at CHEP03[3]: demonstrate the feasibility of interactive analysis of large datasets, set corresponding requirements for grid tools and services, and provide ATLAS a useful distributed analysis environment. By interactive, we do not mean that the user interacts directly with processes running on remote machines, but instead that the system responds promptly, i.e. within a few minutes, to user requests.

The final result is available on this time scale or, if the job is inherently long running or sufficient computing resources are not available, the user can easily monitor progress. In the latter case, the user should have access to partial results and be able to abort the job and free up resources for use by others or to process an amended request.

For large input datasets, this responsiveness is obtained by breaking the dataset up into subdatasets and processing those in parallel. DIAL carries out the splitting of the input dataset and merging of the results so that the user needs to make very little extra effort to harness this distributed processing. Of course, some packaging of the user software is required for remote and/or distributed processing, and so DIAL provides a job definition framework to describe user transformations and datasets.

The C++ and XML specifications of these objects are known as AJDL (Abstract Job Definition Language) and are generic, i.e. not specific to a particular type of data or transformation so that a single DIAL processing system can accommodate requests from different users including those working on different projects.

In the following sections, we describe this framework and the services that DIAL provides to carry out the processing, the monitoring of these services, and the catalogs used to record datasets, transformations and jobs. We describe the system that has been deployed for use in ATLAS and present some performance measurements.

Software

DIAL is mostly written in C++ and the standard distribution kit includes the libraries, executables and header files. The distribution also includes ROOT[4] dictionaries and scripts so most of the DIAL functionality is available at the ROOT command line. A dial shell provides access from a bash command line. The latest release, DIAL 1.30, is built using RedHat[5] Enterprise 3 Linux and is validated on variants of Scientific Linux[6].

A python[7] binding has been created for previous releases and is planned for 1.30. It is packaged separately.

The distribution includes a web service based on gsoap[8] with a GSI plugin[9] allowing authentication and authorization using globus[10] grid proxies. Modules allow this service to be deployed for the purposes described below. Associated client classes facilitate interaction with the services.

All persistent objects have an XML representation that is used to store the objects and for exchange between clients and services.

MODEL

DIAL presents a data-oriented view. By definition, a *job* carries out a *transformation* on a *dataset* to produce another dataset. The output dataset is often called the *result*. A user performing data analysis defines a job and then submits it to a *scheduler* that has responsibility for carrying out the transformation and making the result available to the caller. When deployed as a web service, the scheduler is often called an *analysis service*. We expand on these definitions in the following sections.

Datasets

A dataset is a specification of a collection of data along with some information about the content and organization of the data. Typically a dataset does not carry the actual data but instead holds the *location* of the data, most often as a list of logical or physical file names.

The *content* of a dataset is a description of the nature of the included data. The content is organized as a collection of content blocks, each of which has a label and optional lists of content identifiers and event identifiers. The content label and identifiers can provide users or schedulers with information about the suitability of a particular transformation and to identify the parts of a dataset that are irrelevant to a particular transformation and hence need not be staged for processing.

Datasets are hierarchical, i.e. a dataset may include a list of subdatasets. This structure may arise naturally during construction, as when merging the results for subjobs. Or it may be imposed to provide users or schedulers with hints for splitting or selecting relevant parts of a dataset.

DIAL provides an abstract class Dataset that defines the interface for datasets including access to the above information and means to split and merge. An abstract subclass GenericDataset adds sufficient data to hold the information and provides means for conversion to and from an XML representation. Concrete dataset classes (full types) are required to inherit from the former and typically inherit from the latter, i.e. they adopt its persistent representation. In this case it is possible for a generic scheduler, i.e. one without knowledge of the full type, to manipulate objects of the concrete type, albeit with restricted functionality.

Transformations

Transformations have two components: the *application* that carries out the processing and the *task* that carries data used to configure the application. Processing is carried out in two steps: first the application builds the task and then the input dataset is processed. In the (typical) case of distributed processing, the task is built once (or once for each site or platform) and then each subjob is processed independently using the common task build. If a subsequent job is submitted with the same transformation (i.e. same application and task) and another dataset, the existing task build may be reused.

A task is a collection of named files and each application defines a task interface that specifies the required and allowed names and the nature of the files. The task interface is not yet formalized and so that a scheduler has no means to validate a transformation other than building the task.

An application carries two scripts: *build_task* is used for building and *run* is used for processing. The build script is run in a directory containing the task files and is expected to write the results of the build into that same directory. A job is run in a job directory holding the location of the built task (a directory name in the file *taskdir*) and the input dataset (*dataset.xml*). Either script must return 0 to be considered successful. The run script must also create a file named *result.xml* containing the output dataset.

Both scripts may expect to be run in a minimal environment that includes the usual posix commands, the GNU C++ compiler and pkgmgr[11]. The latter provides an interface for locating other software including DIAL itself.

DIAL defines classes Application and Task that carry the relevant files and provide means for streaming to and from XML.

Job preferences

In addition to the transformation and dataset, a job specification includes job preferences expressed a collection of name-value pairs. These may affect how the processing is carried out but, other than possibly causing a job to fail, should not affect the essence of the result. Results from two jobs run with the same transformation and dataset but different preferences should be equivalent. Examples of sensible preferences are naming conventions for output files, number of subjobs and desired response time.

DIAL provides a class JobPreferences to hold the data and provide streaming to and from XML.

Jobs

Users and schedulers can submit jobs, monitor their progress and interact with them through the interface defined in the class Job. Subclasses provide the means to interact (submit, monitor and kill) with jobs in different batch or workload management systems. In particular, DIAL provides LsfJob to interact with LSF[12] and CondorJob and CondorCodJob to interact with Condor[13] either locally or using Condor-G. Users wishing to make use of other systems can avoid introducing new classes by making use of ScriptedJob that calls a user-supplied script to submit, update and kill jobs. This mechanism has been used to submit jobs to globus gatekeepers and to PANDA[14], the U.S. ATLAS production and analysis system.

The Job class is concrete and provides the persistent representation for all jobs and means to stream to and from XML. In the typical case where a user interacts with a remote scheduler, this is the only visible part of the job and the user must interact with a scheduler to submit, kill or fetch the updated status of a job.

Schedulers

Users normally submit, monitor and kill jobs using a scheduler whose interface is specified in the abstract class Scheduler. At present there are two categories of schedulers: a local scheduler constructs jobs of a particular type using a job creator, and a master scheduler carries out distributed processing using a local scheduler to manage its subjobs. In addition, a client scheduler enables a user to access a scheduler run from a remote web service.

The master scheduler splits the input dataset, creates a subjob for each subdataset and then merges the results from each subjob. The merged result is typically available shortly after the first subjob completes and is then regularly updated as more results come in.

The local scheduler creates job directories on a disk shared with the worker nodes and the jobs are run in those directories. In those cases where the processing is remote from the scheduler, e.g. globus, Condor-G or PANDA, the submitted script makes use of the DIAL command *dial_run_job* which fetches the job description from the analysis service and then runs the job script. Just before termination, these remote jobs tar their run directory and store the resulting archive file in the local storage element for retrieval by the scheduler.

Catalogs

DIAL provides two categories of catalogs: repositories, where objects of the above types (datasets, applications, tasks and jobs) are stored, and selection catalogs, where metadata including a name is associated with selected objects. Upon creation, all objects are assigned unique

64-bit identifiers that are used in persistent references. Objects in repositories are indexed with this identifier and also include a modification time. Except for jobs which have not reached a terminal state, all objects in repositories are immutable making it easy and safe for objects to reside in multiple repositories, e.g. at different sites.

Objects in selection catalogs are indexed by name. Job submission clients typically allow the application, task and dataset to be specified by either name or identifier. Thus a user may use a name to reference the latest version of an object assuming the catalog is updated according to that policy. A user who want to ensure that he or she always get the same object can specify the object by identifier.

DIAL provides MySQL[15] implementations of repositories and selection catalogs and also provides a file-based implementation for repositories. Authorization for MySQL depends on file-resident passwords.

There are repository and selection catalog modules for the DIAL web service and clients for these services have the corresponding catalog interface. In the present deployment all catalogs reside in MySQL and the services access the catalogs using that interface. Client access is provided through web services and requires a globus proxy certificate. Write or update access requires that the owner associated with the object or catalog entry match that associated with the certificate.

USER INTERFACE

All the classes described above are available within C++ and most are also available at the ROOT command line. The latter is the most common way to use DIAL. The interface includes commands to facilitate job submission and retrieval of results with immediate viewing of ROOT histograms and ntuples.

DIAL also provides a command to set up a DIAL environment, e.g. for use inside of applications. It also provides means to start up a DIAL shell, a bash shell with the dial environment. Within this environment, there are commands to construct, retrieve and examine datasets, applications, task and preferences. It is also possible to submit a job.

DIAL web services provide web-based monitors. From any browser a user may view the current list of jobs for an analysis service and examine the properties for each including their lists of subjobs and the properties of those subjobs. One can examine the application, task, dataset, preferences and result associated with any job.

ATLAS DEPLOYMENT

DIAL has been deployed and used for analysis at BNL, the ATLAS tier 1 site in the U.S. The deployment includes analysis and other services, datasets describing data from recent Monte Carlo production, applications for common analysis scenarios, and example tasks for those applications. The DIAL release includes demos for each of the ATLAS applications.

Analysis services

Services at BNL include a unique ID provider, a repository, a selection catalog and a suite of analysis services connected to different processing systems. At present, the best performance is obtained from analysis services that directly access local batch systems. There is a service for each of three effective queues: fast, short and long. For the list of services and up-to-date recommendations on which to use, please see the "ATLAS services" link on the DIAL home page[1].

The fast service is generally the only user of a tuned LSF queue and users can expect jobs to start almost immediately after submission. The dual-CPU worker nodes are typically already running two other jobs and the queue should only be used for very short-running (15 minute) jobs with small memory footprints (100 MB).

The short service makes use of a preemptive Condor queue that will borrow a slot from a machine with a long-running Condor job. Memory is less of an issue but jobs should be short-running (1 hour). There is an inherent Condor latency of 5-10 minutes before jobs will start but there are generally at least a few slots available.

The long service also makes use of Condor preemption but at lower priority and the jobs are subject to preemption by higher priority jobs, e.g. those in the short queue. Long running jobs (1 day) are allowed. Latency depends on usage of the facility and varies dramatically.

At the time of writing the PANDA service had latencies of a few minutes and was only running a few analysis jobs at a time. It is expected this will improve.

There are also Condor-G and globus gatekeeper services that run intermittently at BNL. At present these use the BNL gatekeepers to connect the same queues described above. They offer no gain in performance but demonstrate the feasibility of connecting to remote sites.

Datasets

The AOD (analysis oriented data) data from the last year's Monte Carlo production (Rome data) is available in the form of DIAL datasets. A new production (CSC) has just begun and it is planned that all event data from that production will promptly be made available for analysis with the DIAL services. Preliminary samples produced in the U.S. and few replicated ones are already available.

Transformations

In addition to the generic applications *scriptrunner* and *cxxrunner* that respectively run scripts and C++ programs, four ATLAS-specific applications are available. All make use of the ATLAS framework program Athena. The application *atlasopt* simply runs a job with user-supplied job options (input parameters). *Aodhisto* allows users to additionally provide code to be built inside the standard ATLAS example analysis package. Most popular is *atlasdev* which allows a user to make arbitrary changes to the release and run with those changes provided as a tarball. Finally, *atlasxform* can be used to run the standard transformations used in official data production.

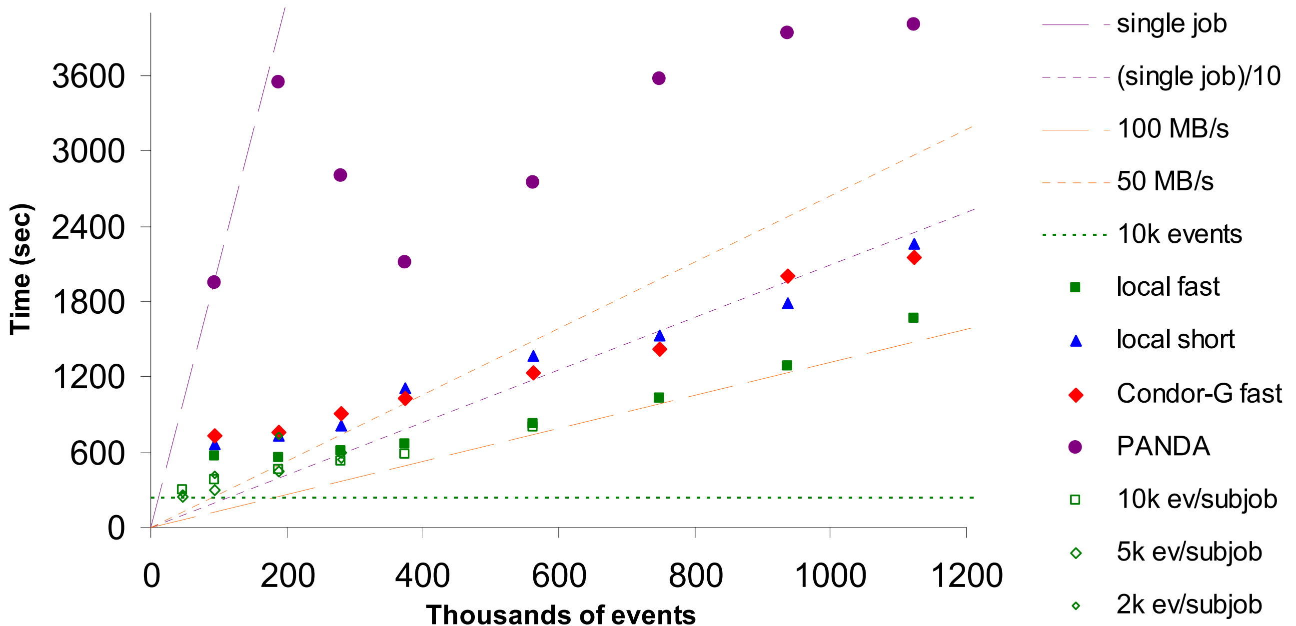

Figure 1: DIAL processing time as a function of the number of events. See text for details.

PERFORMANCE

We constructed a reference dataset for assessing the performance of the system by combining many of the new physics AOD samples available at BNL. This dataset has a total of 1872 files each with 100 events. The total size is 25 GB giving an average of 130 kB per event. The files in this dataset were then copied locally and datasets of varying sizes were created by including up to six copies of each file. The files reside on two NFS file servers.

The *atlasdev* transformation was used with a simple analysis task that opens four containers in each event (truth, electrons, cone jets and jet tags) and fills a couple dozen histograms. Without parallel processing, the reference dataset can be processed in about 130 minutes corresponding to 43 ms/event or 3.0 MB/s. Rough measurement of the actual data transfer rates gives about half of the latter value presumably because there are many other containers (e.g. tracks) that are not being opened.

Figure 1 shows the DIAL processing time as a function of the number of events for various schedulers. For the solid points, the subjob size was fixed at 100 files (10k events) and the number of subjobs varies accordingly from 10 to 113. Results are shown for local fast (green squares), local short (blue triangles), Condor-G submission to local fast (red diamonds) and PANDA (violet circles). For smaller datasets and the local fast queue, results with varying subjob size—100, 50 and 20 events—are shown in open squares, diamonds and triangles, respectively.

Best results are obtained with the local fast (LSF) queue. One million events require about 23 minutes, about 15 times faster than the almost 6 hours required for a single job. There is speedup of a factor of 10 for Condor and Condor-G and a factor of five for panda. The connection to PANDA and PANDA itself are relatively new and we expect significant improvement there.

Direct globus submission was not evaluated because these numbers of jobs put a high load on the gatekeepers that are also used by the production system. Condor-G also uses these gatekeepers but we avoid this problem by using the Condor grid monitor[13].

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support from the BNL ATLAS tier 1 facilities group, many useful comments from members of the ATLAS distributed analysis effort and feedback from the ATLAS physicists who have served as early adopters.

We also acknowledge support from the U.S. DOE through USATLAS and PPDG, PPARC via GridPP and CERN.

REFERENCES

[1] http://www.usatlas.bnl.gov/~dladams/dial.
[2] http://www.cern.ch/ATLAS.
[3] http://chep2004.web.cern.ch/chep2004.
[4] http://root.cern.ch.
[5] http://www.redhat.com.
[6] https://www.scientificlinux.org.
[7] http.//www.python.org.
[8] http://www.cs.fsu.edu/~engelen/soap.html.
[9] http://sara.unile.it/~cafaro/gsi-plugin.html.
[10] http://www.globus.org.
[11] http://www.pp.rhul.ac.uk/~rybkine/pkgmgr.
[12] http://www.platform.com.
[13] http://www/cs.wisc.edu/condor.
[14] http://uimon.cern.ch/twiki/bin/view/Atlas/Panda.
[15] http://www.mysql.com.

Last updated March 9, 2006.