Integration of the ATLAS Tag Database with Data Management and Analysis Components

J. Cranshaw[1], A. T. Doyle[2], M. J. Kenyon[2], D. Malon[1], H. McGlone[2] and C. Nicholson[2]

[1] Argonne National Laboratory, Argonne, IL 60439, USA
[2] Department of Physics and Astronomy, University of Glasgow, Glasgow, G12 8QQ, Scotland

E-mail: c.nicholson@physics.gla.ac.uk

Abstract. The ATLAS Tag Database is an event-level metadata system, designed to allow efficient identification and selection of interesting events for user analysis. By making first-level cuts using queries on a relational database, the size of an analysis input sample could be greatly reduced and thus the time taken for the analysis reduced. Deployment of such a Tag database is underway, but to be most useful it needs to be integrated with the distributed data management (DDM) and distributed analysis (DA) components. This means addressing the issue that the DDM system at ATLAS groups files into datasets for scalability and usability, whereas the Tag Database points to events in files. It also means setting up a system which could prepare a list of input events and use both the DDM and DA systems to run a set of jobs. The ATLAS Tag Navigator Tool (TNT) has been developed to address these issues in an integrated way and provide a tool that the average physicist can use. Here, the current status of this work is presented and areas of future work are highlighted.

1. Introduction

ATLAS, one of the general-purpose experiments at the Large Hadron Collider (LHC) at the CERN, the European Laboratory for Particle Physics, is expected to record raw data at a rate of 200 Hz, giving about 2×10^9 events per year. As set out in the ATLAS Computing Model [1], raw events will have a size of 1.6 MB each. Processing of the raw data into reconstructed events gives Event Summary Data (ESD) files with an event size of about 1 MB; these are then used to produce Analysis Object Data (AOD) events, which contain physics informantion suitable for use in analysis, and which will be about 100 kB each in size. These will generally be stored in files of size 2 GB or larger.

To allow efficient identification and selection of interesting events for users to analyse, ATLAS is deploying an event-level metadata system, which has summary or 'tag' physics data for each event. This tag data can be written in two forms - in ROOT [2] files and as entries in a relational database. The file-based tags are useful as indices to the real data, allowing the position of an interesting event in a file to be located, for example. The relational database-based tags, though, enable the querying of tag data so that a user should be able to select events according to their analysis criteria, find the data files of interest, and be able to go directly to these interesting files and events rather than running their analysis on the whole set of available ESD or AOD.

With a budget of 1 kB per event, such a relational database would host about 2 TB of new data each year. While this is small relative to the overall scale of ATLAS data, having an efficient

and scalable database of this size is still a challenge. A series of performance and scalability tests are being performed, seeking the best design for such a database, and the results of some of these tests can be found in [3].

Another challenge is the integration of such a database with the other software components used by ATLAS, in such a way as to let users select the events and files of interest and then run their analysis on these events at an appropriate grid site, sending jobs to the data wherever possible. This paper first gives a brief description of the Tag Database itself, then of the Distributed Data Management and Distributed Analysis systems used by ATLAS, highlighting the issues involved in integrating the Tag Database with these systems. The development of a solution, the ATLAS Tag Navigator Tool (TNT) is then described, some test results are presented and future plans are outlined.

2. The ATLAS Tag Database

The ATLAS Computing Model [1] describes a multi-tiered system in which CERN is a central 'Tier-0' site, with regional 'Tier-1' centres around the world, each of which has a number of more local 'Tier-2' sites associated with it. It is planned to have copies of the file-based tags on all tiers. For the relational tags, i.e. those stored in a relational database, there will be a central global database at CERN, hosted on Oracle. To ease the load on this central service, the database will be replicated to various other Tier-1 and Tier-2 sites. Both Oracle and MySQL may be used as database backends, depending on the capabilities at the sites. The exact distribution model for the Tag Database is still under discussion, however.

The central Tag Database is generated from the file-based tags, which are produced alongside the Analysis Object Data (AOD) in the ATLAS production system. Both types of tag data are therefore exactly equivalent, the advantage of a database being that it can be queried. The content of the tags can be divided into six types of attribute. These are:

- Event quantities - attributes that apply to the whole event, such as run number, event number, luminosity and so on
- Data quality - the status of the various detectors, with a boolean 'Good for physics' if all were satisfactory
- Physics objects - electrons, muons, photons, taus, jets and their attributes
- Physics or Performance Group attributes - space for each physics group to define its own attributes
- Trigger information - for both low and high-level triggers.
- Pointers to event data - references to the AOD, ESD and RAW data files which contain the event, software version used and so on

A full list of the current tag attributes may be found in [4], although some of them are likely to change over time as real data-taking begins and user access patterns become apparent. Users may perform queries using any of these attributes to find the events they are interested in. Particularly relevant to this paper is the AOD reference, which is part of the collection information. In the relational database, this is linked to the GUID (Globally Unique Identifier) of the AOD file which contains the event, and this GUID may be returned by a query.

A series of Tag Databases have been deployed at CERN using Monte Carlo data, to allow testing of the database by developers and physicists before the LHC is turned on and real data-taking begins. The largest of these test databases was 1 TB, which was limited by the database resources available. The most realistic database constructed to date, in terms of tag content, contains (at the time of writing) 1.9 GB of data and 1.5 GB of indices, which, although much smaller than what will be required when data-taking starts, allows testing of the functionality and interactions with external components. This is the database which was used for the tests

presented in Section 5. A web query interface has been developed [5], allowing users to browse the database in an intuitive way and to download the results of their query as a ROOT file.

3. The ATLAS Distributed Data Management and Distributed Analysis Systems

ATLAS makes use of three distinct computing grids to process and analyse its data: the LHC Computing Grid (LCG), the Nordic DataGrid Facility (NDGF), and the Open Science Grid (OSG). This is a complex environment in which to manage data and perform distributed analysis, and so a brief overview is given here of some of the systems developed to handle this, with which the Tag Database must interact.

3.1. The Distributed Data Management System

The Distributed Data Management (DDM) system, the implementation of which is known as Don Quijote 2 or DQ2 [6], was developed to handle the movement and cataloguing of ATLAS data files across these three grids. Each grid has its own set of middleware for low-level file cataloguing, storage and movement, so DQ2 provides a common interface for these services. It relies on the concept of datasets. A dataset is a set of files, with certain metadata associated with it such as version number, location, whether more files can be added or not, and many others. The dataset is the unit of data transfer in the DDM system; users are not able to handle files at the individual level in DQ2, although they can do so using the underlying grid tools.

Datasets are transferred between sites using a mechanism of *subscriptions*. Each participating site has a set of *site services* running, and when a site is subscribed to a particular dataset, the site services are responsible for transferring that dataset and keeping it up-to-date should new files be added. A set of central catalogues keeps track of the files in each dataset, the dataset locations, identifiers and subscriptions.

3.2. The Distributed Analysis System

The ATLAS Distributed Analysis system [7] aims to make the computing resources of the three grids available to physicists for their analysis, while hiding the complexity which is involved. There are several tools which are being developed for this, including PANDA [8] and GANGA [9]. PANDA is a job submission and management system developed primarily for OSG but now extended to include LCG sites. GANGA is a user interface for job definition and management on the grid, with a plugin architecture which allows it to run on various backends, developed by ATLAS in conjunction with the LHCb experiment. As the interfacing of the Tag Database with the Distributed Analysis tools was done through GANGA (Section 4.3), a short description is given here.

In GANGA, everything is constructed around objects known as GANGA jobs. Each GANGA job must have defined an application to run and the backend system on which to run it. Most jobs will also have an input dataset of files to read and an output dataset to contain the results. Jobs can also have a splitter defined, which gives a rule for dividing the job into a set of smaller jobs which can be run in parallel, and a merger, which gives the rule for re-combining the output from the sub-jobs. Each of these components of a job (application, backend, input, output, splitter and merger) can be implemented in various ways, as different plugins. ATLAS users, for example, can use the `Athena()` application, which gives access to Athena, the ATLAS analysis framework. Examples of available backend plugins are `Local()`, where the job is run on the local host; `LCG()`, where it is submitted to LCG; and `NG()`, where it is submitted to the NDGF. All the ATLAS-specific parts of GANGA are kept in a package called GangaAtlas.

Users may interact with GANGA through its own enhanced Python shell, known as CLIP; through batch scripts; or through a graphical user interface. When all the necessary inputs to a job have been defined and the job submitted, GANGA then performs all the necessary monitoring and handling of output.

3.3. Issues involved in interfacing with these systems

In order to integrate the Tag Database with the wider ATLAS infrastructure, and the above components in particular, there were several issues to be considered. One of the main problems was that while the DDM system emphasises the use of datasets and does not work at the file level, the Tag Database has no knowledge of datasets and only contains references to data files. To use the Tag Database to locate files with events of interest and then use these in an analysis therefore requires some bridging between the two systems in order to find which datasets contain these events.

In interfacing with the distributed analysis system, it was decided to integrate first with GANGA rather than PANDA or other tools, due to GANGA's modular design and its plans to include access to PANDA through another plugin. It was found that while GANGA already had functionality for analysis using file-based tags, using relational tags was not supported and it was necessary to develop a new GANGA plugin for that.

4. Design and Development of the Tag Navigator Tool

In view of the systems described above, a tool was developed to allow use of the Tag Database in an integrated way. This has been named the Tag Navigator Tool (TNT); it was developed first as a standalone set of Python scripts and then integrated with GANGA as a plugin. In this section, an example is first given to illustrate the motivation for the design. Descriptions are then given of both the standalone version and the GANGA plugin.

4.1. An example use case

The chief use case for TNT is that of a physicist, wishing to query the Tag Database to find some interesting events and then have some analysis run on those events using Athena, without having to know about where these events reside. The steps required would be:

(i) Physicist decides on query, perhaps refined using the database web interface [5], and prepares a file with job options for Athena

(ii) Query, job options and other parameters are submitted to TNT

(iii) TNT queries Tag Database with given query and gets list of matching events

(iv) TNT finds which files and datasets these events belong to, and where they are

(v) TNT prepares a series of LCG jobs, one for each input file

(vi) Jobs are submitted to LCG and TNT waits until they are all done

(vii) Output is either returned to the user or registered as a new dataset, accessible with DQ2, according to user's preference.

A variant on the above case would be when the user has already performed their query on the database and has the matching events available to them in a ROOT file. The web interface to the Tag Database, for example, allows this to be done. In that case, steps (i) and (iii) should be omitted from the use-case above. The implementation of TNT described below is based on these use cases.

4.2. Standalone version

The first version of TNT was developed as a set of shell scripts, which were later translated into Python. As it runs independently of other distributed analysis software, it is here called 'standalone TNT'.

This implementation is a wrapper around various existing tools. These are: the utilities developed by POOL [10] to handle "Collections", of which tags are an instance; grid job submission and management tools; and DQ2 tools for handling datasets. A diagram illustrating the various components and their interactions is shown in Figure 1.

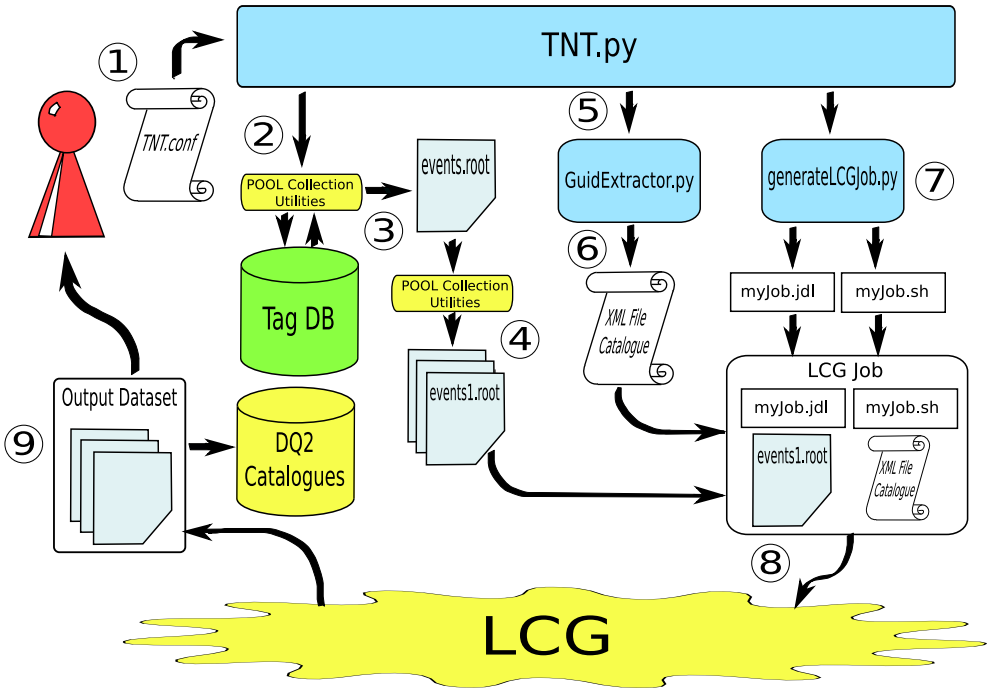

Figure 1. Components and their interactions in standalone TNT

In the diagram, the python components (`TNT.py`, `GuidExtractor.py`, `GenerateCatalogs.py` and `generateLCGJob.py`) are those specific to TNT. The numbers indicated in the description below refer to those on the diagram. Following the steps defined in the use case in the previous section, all the user's query-, Athena- and grid-related parameters are first defined (1) in a configuration file, called `TNT.conf` in the diagram. The main executable, `TNT.py`, is then called and the process is started. TNT uses a POOL-supplied utility to pass the user's query to the database (2) and copy the relevant events locally as a ROOT file (3). In the diagram, this is called `events.root`. A customised version of one of the POOL utilities was written to iterate through this collection of events and split them into a number of sub-collections (4), according to which AOD file they belong to. There can also be a minimum number of events specified by the user, so if the number of relevant events in one file is lower than the minimum, events from another file or files will be included in the same sub-collection until the minimum is reached. Files are not split between sub-collections, however; all the splits are on file boundaries.

Next, the `GuidExtractor` script is called (5), which addresses the gap between the dataset-oriented DDM system and the file-oriented Tag Database. It looks up the relevant DQ2 catalogues, via their API, first to map the GUID of each AOD file to the latest version of a dataset in which it is contained, and second to list the files in that dataset and match the correct Logical File Name (LFN) to the GUID. After this, an XML-based file catalogue is generated for each sub-collection (6), cataloguing the GUID, LFN and Physical File Name (PFN) of where the file will be when it is copied to the worker node. This is later sent with the grid job so that processes running on the grid worker nodes can locate the correct files. For each sub-collection, the `generateLCGJob.py` script then produces 2 files: the executable script which will be run on the worker node, and the JDL file with the correct parameters for submission to LCG (7).

Each job is then submitted (8), taking with it the sub-collection of tag events corresponding to the AOD it is to analyse, the Athena job options file it is to run and the XML file catalogue which will enable it to locate the correct AOD files. The LCG Resource Broker decides where

the job should run; TNT polls the Resource Broker to check job status until all the jobs have finished running, resubmitting failed jobs (unless the failure was failure of the analysis). After a job has finished running, any output files can either be returned to the user, or registered as a new dataset in the DQ2 catalogues (9).

In the case where a user already has a file with tags they want to use, this can simply be done by setting some parameters appropriately in the configuration file.

The main limitation to standalone TNT at the moment is the reliance on LCG as the grid backend - there is no option to use either of the other grids or a local machine or batch farm.

4.3. GangaTnt plugin

As it is expected that much ATLAS analysis work will be done through GANGA, it is sensible to include access to the Tag Database through GANGA as well as in a standalone version. This also makes use of the existing job submission and handling infrastructure developed in GANGA, and removes the dependency on LCG, as GANGA backend plugins to other grid types can be used as they become available. The greater maturity of the GANGA project and growing familiarity of users with it would also facilitate uptake by users. The plugin, known as GangaTnt, is therefore expected to be the more useful instantiation of TNT and its use is encouraged in preference to the standalone version.

The GangaTnt plugin was prepared by adapting the relevant parts of TNT while leaving out the parts for which GANGA already provided functionality. In GANGA, there is a family of `Splitter` classes, which tell Ganga how to split up a job into a number of sub-jobs. Much of the development of GangaTnt consisted of writing a new `Splitter`, `TNTJobSplitter`, which included the Tag database lookup and then the splitting of the returned events (and hence the jobs) along AOD file boundaries, in the same way as it is done in the standalone version.

The main components of GangaTnt and their interactions with the Tag Database and GANGA are shown in Figure 2. The user may interact with GANGA through its own command-

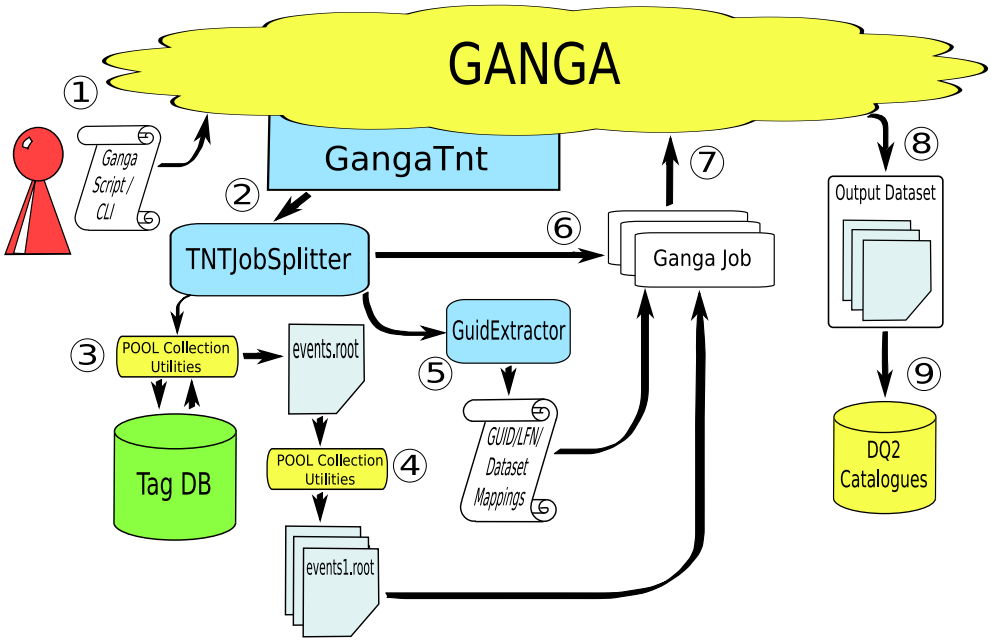

Figure 2. Components and their interactions in GangaTnt

line interface or in batch mode via a script (1), in the same way as for any GANGA job. The

`TNTJobSplitter` class is instantiated (2) and the selection of events made from the database (3) and returned to the local area. The set of events returned is split up (4) and `GuidExtractor` called to get the appropriate list of GUIDs. As in the standalone case, it then looks up DQ2 (5) to match the GUIDs to the datasets to which they belong and the corresponding LFNs. The dataset locations are also found and used to tell GANGA the best sites to send the jobs. The splitter then prepares the correct number of jobs (6), which are then submitted to GANGA (7). GANGA then takes care of submitting them to the correct backend, according to the user's choice. The default, which should be adhered to wherever possible, is to send the sub-jobs to the sites where the AOD files are already present, to avoid unnecessary file transfers across the grid. GANGA then monitors their status, collects the output, and either returning it to the user or, as is more usual, registers the output files in a new dataset registered in DQ2 (9). They can then be manipulated using GANGA, DQ2 or native grid tools, as required.

If a user has already performed a query and got some events in a ROOT file, this can be analysed using the existing functionality GANGA has for ATLAS tag files. GangaTnt is therefore complementary to the rest of the GangaAtlas package; together, they can cover all sorts of analysis where there is pre-selection using tags.

5. Some Performance Measurements

TNT, in its standalone implementation and as GangaTnt, has successfully been tested using the existing Tag Database at CERN. To provide some preliminary measurement of the performance of GangaTnt, some simple tests have been carried out, comparing an analysis with pre-selection using tags to analysis performed just on the AOD. A series of tests was first conducted to give an initial understanding of the performance of using tags independently of the wider distributed analysis framework, followed by some tests using GANGA in the LCG environment.

For all the tests described below, the analysis used was a simple reconstruction of the Z mass from the $Z \rightarrow e, e$ process, with events which have been loaded into the central Tag Database at CERN. Each result shown in the plots is the average of three measurements.

5.1. Tests in the Local Environment

First, the analysis was done on a single AOD file, initially without using tags and then with event selection using the corresponding file-based tag. Both files were present on the local disk. For this single file, the time taken to run Athena on the whole file was measured, selecting events electron $p_T > 20GeV$ and $|\eta| < 2.5$, which is roughly 10% of the sample. The same analysis was then performed using the tag file to pre-select a varying percentage of the events, which were then analysed in their AOD form. The results are shown in Figure 3. The CPU time is shown rather than the real time taken, to avoid any contribution to the real time from competing processes on the host machine. The horizontal line, showing the time taken without tags, has no data points because no percentage pre-selection is being made - it is there as a baseline for easy comparison. This time does not vary significantly as the AOD selection is changed in any case. The graph shows that if the percentage of events selected using tags is less than about 60%, using tags can give a significant improvement in analysis time. With tighter selection, the reduction in time increases, so for a 10% selection the analysis time is reduced by about 50%. For very loose selections where 60-100% of the events are passed by the tag, the overhead from navigating between tag and AOD makes it slower than doing the analysis without tags.

Next, the time taken for the same analysis was measured for an increasingly large set of input files and, correspondingly, events. Each file contained about 4-5000 events. Again, all files were present on local disk. The analysis was done first without using any tag pre-selection; second, using file-based tags; and third, using the Tag Database. While the same input files were used for the first two sets of analysis, the nature of the Tag Database meant that the finest granularity available was at the dataset level. That is, each dataset loaded into the database

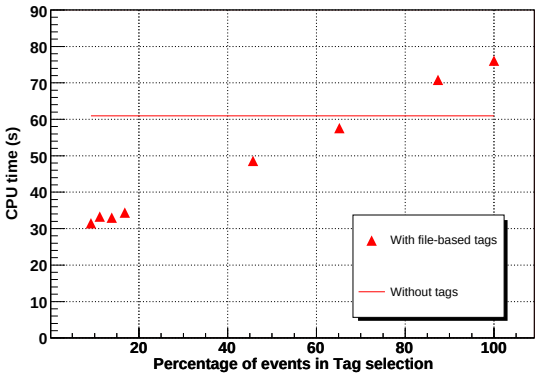

Figure 3. Analysis time with varying percentage of pre-selected events

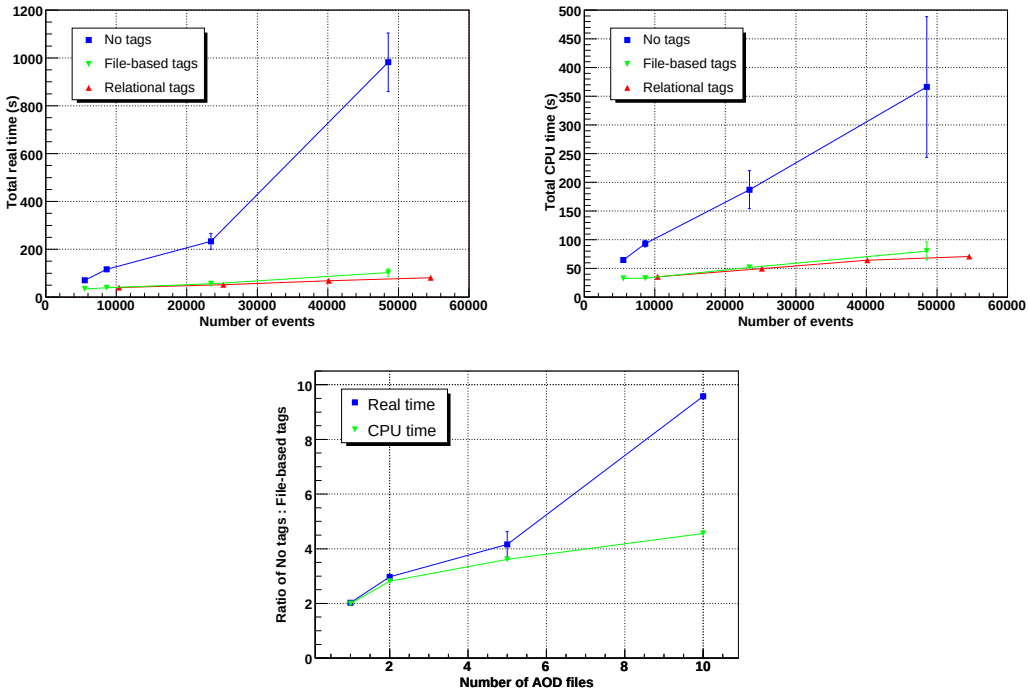

Figure 4. Analysis time with varying number of AOD files. Real time (top left), CPU time (top right), and ratio of times with and without file-based tags (bottom).

has a different run number and thus can be found by a simple query, but there is no way to query for a particular file. Thus, while most of the files used were the same as in the previous tests, some were slightly different. The first two plots in Figure 4 are therefore shown in terms of increasing number of events. The third plot, which shows the ratio of times with and without using tags, only uses the file-based tags and is shown in terms of increasing number of files. These plots show that as the number of input events increases, the performance gained by using tags increases, so with 10 AOD files and almost 50000 events, using tags is over 4 times faster in CPU time alone. In general, this increase is linear with the number of events, although there

may be some non-linearity in the real time taken without tags. More data is needed to quantify this and to explore why there is this change in the ratio of analysis speeds. As file sizes are expected to increase, the results at higher numbers of events are more realistic, and extension of the tests to even higher numbers is needed. There was little significant difference in performance seen between the two kinds of tag, although further work is needed to compare these in more detail. As the database is resident at CERN, for example, and the tests were performed on a machine in the CERN LAN, it would be interesting to see the impact of queries over a WAN.

5.2. Tests in the Distributed Analysis Environment

Having performed the previous tests in a local environment, some tests were then made running this analysis in the distributed analysis environment through GANGA. Two files from a single AOD dataset were used as input, and GANGA was set to send the jobs to LCG sites where this AOD dataset was already present. For each job, two time measurements were taken on the worker node on which it ran: the time for the script to set up, including the fetching of any files if necessary (the *setup time*), and the time for the analysis to run (*analysis time*). The same cuts were made as in the previous section, electron $p_T > 20GeV$ and $|\eta| < 2.5$.

First, the time for the analysis without AOD was measured, then the time for the analysis using the tag file datasets as input to GANGA. No job splitting was done. Finally, the GangaTnt plugin was used. In this case, as explained in Section 4.3, the query to the Tag Database is performed from the local host before the jobs are submitted to LCG. A minimum number of events per job was set so that each job processed both the input files, rather than splitting into parallel jobs, to allow direct comparison of the results.

The results are shown in Figure 5. Looking first at the setup times on the worker node, it is

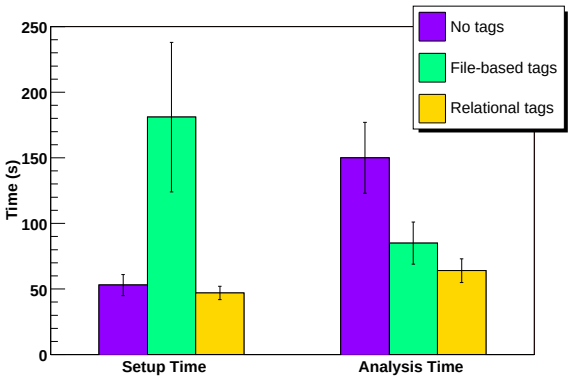

Figure 5. Comparison of setup and analysis times on worker nodes with Ganga

seen that the time without tags and the time with relational tags is very similar, while the time with file-based tags is much higher. However, this is due to the fact that at present, the tag files are not being replicated together with AOD files. This means that while GANGA sent the jobs to sites with the AOD files, the tag files were not present and were therefore downloaded to the worker nodes from a remote site. It is expected that in future, the tag files will be present at all sites and this loss of time will not occur, so all setup times should be similar.

Second, looking at the analysis times, it is clear that using the tags was about twice as fast as doing the analysis without tags. Again, there is little difference between the two tag models. There may be a small increase in speed with the GangaTnt model, which could be due to the selection having been made previously on the database; however, this is not a significant

difference in terms of the overall process. Separate measurements of the queries executed on the database show query times of about 2 seconds; this, together with other lookup overheads in GangaTnt, counterbalance any performance gain on the worker node.

In this simple test, the gain in analysis time from using tags gave only a small gain in the total time taken from job submission to completion, because there can be long wait times between job submission and beginning to run on a worker node. However, with larger analyses, the results in the previous section show that the impact of using tags will be considerably higher.

In summary, these results show that the GangaTnt model and the standard GANGA model of using tags in analysis perform equally well in general, although the GangaTnt approach has advantages in the case where the correct tag files are not present at the site the job is running at. Both are useful in different situations: the ordinary GANGA method if the user knows the tag files they need and knows they are at the site, or already has them locally, and GangaTnt when a query to the relational database is needed or the files are not widely distributed.

6. Conclusions and Future Work

TNT and GangaTnt have been developed to enable integration of the ATLAS Tag Database with the distributed data management and distributed analysis frameworks. An initial series of tests have been performed which show that using tags for pre-selecting events for analysis gives about 50% improvement in analysis time for a 10% selection on a single file of about 5000 events; this improvement increases as the number of input events is increased. Within the distributed analysis framework, using standard GANGA Tag analysis and GangaTnt give similar gains in performance compared to analysis without tags, and both are useful in different circumstances.

Further work is required to better understand the effects of using tags with large numbers of input events; the effect of file I/O; and the differences between file-based and relational tags. A more extended series of tests to this end is planned. In terms of development, the ATLAS Tag Database will continue to grow as data is added, and GangaTnt will continue to evolve as feedback is given by users and as the data management and analysis components develop in the approach to LHC data-taking in 2008.

Acknowledgments

The authors would like to thank GridPP and ATLAS eScience. Argonne National Laboratory's work was supported by the U.S. Department of Energy, under contract DE-AC02-06CH11357.

References

[1] The ATLAS Computing Model. Technical Report CERN-LHCC-2004-037/G-085, CERN, January 2005.

[2] R. Brun and F. Rademakers. "ROOT - An Object Oriented Data Analysis Framework", *Nucl. Inst. & Meth. in Phys. Res. A* **389** (1997) 81-86. See also http://root.cern.ch/

[3] J. Cranshaw, L. Goossens, D. Malon, H. McGlone and F. T. A. Viegas. "Building a Scalable Event-Level Metadata System for ATLAS", *Computing in High Energy and Nuclear Physics (CHEP)*, Victoria, Canada, September 2007

[4] https://twiki.cern.ch/twiki/bin/view/Atlas/TagForEventSelection

[5] http://atlas.web.cern.ch/Atlas/GROUPS/OPERATIONS/dataBases/TAGS/tag_browser.php

[6] M. Branco, D. Cameron and T. Wenaus. "A Scalable Distributed Data Management System for ATLAS", *Computing in High Energy and Nuclear Physics (CHEP)*, Mumbai, India, February 2006

[7] D. Liko. "The ATLAS Strategy for Distributed Analysis on several Grid Infrastructures", *Computing in High Energy and Nuclear Physics (CHEP)*, Mumbai, India, February 2006

[8] K. De, T. Wenaus et al. "Panda: Production and Distributed Analysis System for ATLAS", *Computing in High Energy and Nuclear Physics (CHEP)*, Mumbai, India, February 2006

[9] K. Harrison, C.L. Tan, D. Liko, A. Maier, J. Moscicki, U. Egede, R.W.L. Jones, A. Soroko and G.N. Patrick. "Ganga: a Grid user interface for distributed analysis", *Proc. Fifth UK e-Science All-Hands Meeting*, Nottingham, UK, September 2006

[10] D. Duellmann on behalf of the POOL project. "The LCG POOL Project - General Overview and Project Structure", *Computing in High Energy and Nuclear Physics (CHEP)*, San Diego, USA, March 2003