A Data Skimming Service for Locally Resident Analysis Data

J Cranshaw[1], R W Gardner[2], J Gieraltowski[1], D Malon[1], M Mambelli[2] and E May[1]

1. Argonne National Laboratory, Argonne, IL 60439, USA
2. Enrico Fermi Institute, University of Chicago, Chicago, IL 60637, USA

E-mail: marco@hep.uchciago.edu

Abstract. A Data Skimming Service (DSS) is a site-level service for rapid event filtering and selection from locally resident datasets based on metadata queries to associated "tag" databases. In US ATLAS, we expect most if not all of the AOD-based datasets to be replicated to each of the five Tier 2 regional facilities in the US Tier 1 "cloud" coordinated by Brookhaven National Laboratory. Entire datasets will consist of on the order of several terabytes of data, and providing easy, quick access to skimmed subsets of these data will be vital to physics working groups. Typically, physicists will be interested in portions of the complete datasets, selected according to event-level attributes (number of jets, missing Et, etc) and content (specific analysis objects for subsequent processing). In this paper we describe methods used to classify data (metadata tag generation) and to store these results in a local database. Next we discuss a general framework which includes methods for accessing this information, defining skims, specifying event output content, accessing locally available storage through a variety of interfaces (SRM, dCache/dccp, gridftp), accessing remote storage elements as specified, and user job submission tools through local or grid schedulers. The advantages of the DSS are the ability to quickly "browse" datasets and design skims, for example, pre-adjusting cuts to get to a desired skim level with minimal use of compute resources, and to encode these analysis operations in a database for re-analysis and archival purposes. Additionally the framework has provisions to operate autonomously in the event that external, central resources are not available, and to provide, as a reduced package, a minimal skimming service tailored to the needs of small Tier 3 centres or individual users.

1. Introduction

The Large Hadron Collider (LHC)[1] is being built at the CERN laboratory, near Geneva, Switzerland. It is a proton-proton collider with a center of mass energy of 14 TeV. Over the next 10 years this tool should provide sufficient collisions to provide strong tests of the Standard Model and its various extensions such as SUSY. Two general purpose detectors (ATLAS[2] and CMS[2]) as well as two special purpose detectors (ALICE[2] and LHCb[2]) are being installed to measure the properties of these collisions which can then be used to calculate physics results.

The amount of data produced by the detector is large both because of the number of channels and because of the event rate. At ATLAS the raw data from the detector corresponds to ~2 MB per event with an event rate of 200-500 Hz. The raw data will then go through a rather standard series of reconstruction and particle definition steps, which will yield Event Summary Data (ESD), Analysis Object Data (AOD) and Derived Physics Data (DPD). Section 2 describes these data products, but their exact definitions and uses are beyond the scope of this paper, and are discussed in other documents, such as the ATLAS Computing TDR[3]. This structure gives a total data production of O(10 PB/yr)[1]. Data sizes like this require a navigational infrastructure. They also benefit from the parallel processing capabilities of the Grid, although the use of distributed systems can impose a significant overhead.

Unlike some grid applications involving computationally-intensive calculations, here the emphasis is more upon throughput of data and scalability. This paper presents in Section 3 a tool, called Data Skimming Service, that attempts to use navigation tools and to provide an appropriate processing solution to improve user access to the data needed to quickly extract physics results. It further describes a prototype that has been deployed at the Midwest Tier2[4] at the University of Chicago. Finally Section 4 compares the approach presented in Section 3 with other solutions used in ATLAS for data analysis or reconstruction.

2. ATLAS Data management and TAG information

As noted before ATLAS will produce large amounts of data with various levels of processing: RAW (2MB/event), ESD (1 MB/event), AOD (0.2 MB/event), DPD. These data will be distributed across a multi-tier group of data centers with the higher numbers having progressively lower levels of facilities and support. The Tier 0 facility is CERN, while Tier 1 facilities are national facilities used by ATLAS and Tier 2 facilities are regional. Tier 3's also exist to exploit opportunistic resource access or funding opportunities.

Not all data will be present at all Tiers nor will all Tier resources be available for user analysis. The ATLAS computing model describes the Tier 0 facility as essentially the back end of the DAQ, which provides fully reconstructed events in a timely manner. It will contain a full copy of the RAW data as well as full copies of the first pass ESD, AOD, DPD. There will be approximately 10 Tier 1 facilities. The resources of Tier1 will be used primarily to support reprocessing of RAW data and dedicated productions approved by physics groups. They will contain partial copies of the RAW data, partial copies of the ESD data, and full copies of the AOD data (including post first pass). There will be roughly three times as many Tier 2 facilities as Tier 1 facilities. These will be used extensively for Monte Carlo productions. They will contain copies of any Monte Carlo data produced (RAW) and as large a copy of the AOD data as can be accommodated by site resources. User analyses are expected to concentrate on using AOD and DPD with access to the RAW and ESD data controlled by computing operations and the physics groups.

2.1. ATLAS data management

Both raw and processed event data are stored in files. In order to distribute this data ATLAS has set up a Distributed Data Management system (DDM), called Don Quijote (DQ2)[5], which supports grouping of files into datasets for efficient transfer. To trigger data movement the sites register subscriptions to the desired datasets. The DDM then tracks where copies of the datasets exist and is then free to use internal optimizations to deliver the data to the site that subscribed to it. Any system that needs to access the global ATLAS data store for files will then use the DDM to locate them.

2.2. Event tagging and metadata

The proposal for a metadata system for ATLAS is described in D. Costanzo et al. [6]. 'Tag' data, often referred to simply as TAG, is metadata used for event characterization and selection. The metadata consists of tags, name value pairs, stored separately from the event data but with a reference to the event data itself.

The TAG uses the LCG POOL Collections [7] to provide a method to navigate to the events within the files in the data store. A set of utilities for accessing the collection data is also provided, although ATLAS has also extended these for ATLAS-specific applications. LCG POOL Collections also provide for multiple persistent storage mechanisms. In our case the two mechanisms of interest are relational databases and ROOT[8] files. In POOL, references to events contain the file ID, object ID, and position within the file. For the TAG the object ID is always the DataHeader for the event. These tools support multiple data transformations both with and without selection:

- AOD -> AOD based on TAG (skim off a subset of events)
- TAG -> TAG based on TAG (store a selection for later use)
- TAG -> new TAG with new attributes (store a selection with new attributes for later selection)
- AOD -> new TAG with new attributes

Combinations of these transformations are also possible.

The TAG data are first written to files during the same stage as AOD and DPD production. It is then loaded into a global database for browsing by users, tuning of selections, and input to analysis jobs. Whenever convenient, local databases can replicate the global one or directly load TAG data[9] from files. The metadata in the TAG falls into several rough categories

- Event ID (run number, event number, etc.)
- Quality (calorimeter good, good for SUSY, etc.)
- Trigger (which triggers fired)
- Physics (number of electrons, missing Et, etc.)

The content of the TAG was reviewed in early 2006 and a solid set of quantities was chosen, which can be evolved as needed [10].

In order to access the event data, the TAG uses the LCG POOL Collections [7] to provide a method to navigate to the events within the files in the data store. In general the TAG contains references to all previous stages of process, e.g., AOD, ESD, RAW. The default reference returned is to the AOD, but users can choose the other stages with a simple switch, although access to ESD and RAW is normally restricted by the data management groups. Integration of the TAG navigation with the DDM is described in C. Nicholson et al. [11].

2.3. Event selection

Much of the initial work of a physics analysis is event selection. This begins at the detector with the trigger system. After this the events will be streamed into roughly 5 streams based on related triggers. The latter is not really an event selection process but rather a sorting for efficient event placement. The TAG is meant to allow the user to then select certain events within a stream to use as input to a job. The user can then run analysis-specific code and create an analysis specific dataset (where dataset here is used to simply mean an event list, not a group of files). It should then be possible to publish this dataset for use by others.

A standard user pattern is to cull this event list as quickly as possible to include only events useful for a specific analysis. This may actually include the production of several event lists on several streams or across streams. But once the event lists are generated, users normally want to run over that dataset multiple times on resources where they can reliably schedule their analysis. Data movement is one of the most problematic activities in a distributed system; therefore the users want to limit it. They have several options for reducing the size of the dataset.

- Skimming: The extraction of events of interest from the data store.
- Slimming: Storing a subset of the data classes rather than a full copy of the event.
- Thinning: Limiting the objects based on usefulness for their physics

Of these options, skimming seems the easiest to generalize and a natural first place to begin for providing a data service.

3. Data Skimming Service

The Data Skimming Service (DSS) is a service that could be deployed at ATLAS Tier2s, allowing scientists to easily select some events and reducing the amount of data on which they run their analysis. At the moment the service exists as a prototype installed on a host at the Midwest Tier2 (MWT2_UC). It can be used issuing shell commands or using its Web interface. The Web interface has been chosen because it is accessible using a normal Web browser: it is a very common technology, it is a GUI that guides the user and it is well-known interface: most users know how to interact with web pages. Future versions will include also an API (python functions and a Web service or XML-RPC) to allow interactions with other programs that will be able to use DSS to retrieve skimmed datasets to use in their workflow. The term 'user' in the remainder of the paper refers to both a human user and an automatic agent, e.g. a program, interacting with DSS.

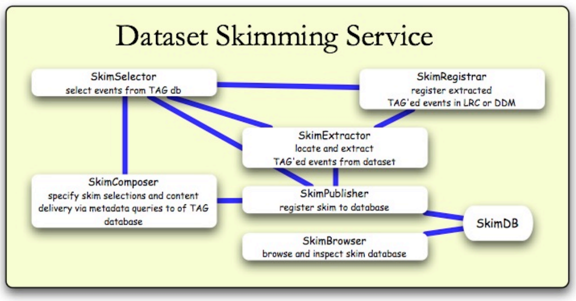
Dataset Skimming Service
SkimSelector
select events from TAG db
SkimRegistrar
register extracted
TAG'ed events in LRC or DDM
SkimExtractor
locate and extract
TAG'ed events from dataset
SkimComposer
specify skim selections and content delivery via metadata queries to of TAG database
SkimPublisher
register skim to database
SkimDB
SkimBrowser
browse and inspect skim database

Figure 1. DSS components and interactions.

3.1. Design of DSS

DSS is composed of modules that each provide specific functionality as shown in figure 1. They work together using core software and a database that contains most of the state of the service. An exemplar use-case may be formulated as follows:

- production of 'tag' information analyzing the AOD;
- loading of 'tag' information in a relational database;
- browsing of the database to get familiar with the data and formulate a query;
- selection of the events that match all the desired requirements;
- extraction of the events from the original AOD datasets;
- copy to a storage element the files containing the desired events;
- registration in the ATLAS data management system of the skimmed dataset to make it available on the Grid;
- execution of the desired analysis/processing on the skimmed dataset.

The user is free to interact with the service in many different ways: joining and leaving the workflow above or repeating some of the steps. For example, some users may prefer to produce the event selection somewhere else and use the skimming service only for the extraction; some others may take the event selection and extract the events using a distributed analysis tool. After completing the sequence above it is possible to produce 'tag' information of the skimmed dataset and restart the whole sequence, refining the first selection. The rest of the section will describe the components in the order in which they are encountered in the example workflow above. To have a uniform description each subsection ends with 3 paragraphs the first describing the inputs of the module (It uses…), the second the outputs produced (It provides…) and finally the last on describing the system and software requirements (It requires…).

3.1.1. *SkimDB (SkimDatabase)*. Contains the state and histories of active and archived skims. It does not appear explicitly in the workflow above but it is involved in tracking the state of most of the DSS activity. It collects bookkeeping and provenance information to document how a skim was obtained, to retrieve its products and to know its status. It is actually part of a bigger database containing all the persistent information used by the service; this includes status information, characteristics of the available computing and storage elements, description of available 'tag' information (TAG databases). Since all the modules of DSS interact with this database updating some status information, the interactions are not mentioned explicitly in the next paragraphs.

It is updated each time a new skim is requested or a significant step in its production completed.

It provides information about the skims to all other components.

It requires a database backend (e.g. MySQL or sqLite).

3.1.2. *Data moving utilities (DMU)*. This is another component used mainly behind the scenes. It moves files reliably to and from storage elements. Besides copying the files, it compares file sizes, and checksums if possible, to ensure the correctness of the copy. It further provides a timeout and a retry mechanism to improve reliability. It supports different protocols including plain file copy, dCache[12], xrootd[13], ftp/gsiftp, HTTP, SRM[14]. A user client tool provides a command line interface, and the same library is used by the Panda pilot[15] for data transfer to/from the local storage element.

It uses information about storage elements and the dataset or the files that are being copied.

It provides a copy of the file.

It may require dCache or Globus[16] client tools depending upon where are located the files that are moved.

3.1.3. *Skim composer*. Iteratively compose skims, specifying event cuts and output content. Some standard/predefined skims speed up the most common user interactions. A user interface allows the user to enter parameters for custom skims. Some exploratory queries present preliminary results (e.g., count the number of records, plot some basic quantities) and help the user to characterize the data. A pre-defined set of plots can summarize the distribution of important parameters. This component could either be an interactive tool or a simple browser of the database containing tag information.

It uses the database containing tag information, and user-provided input.

It provides some representation capturing the selection (e.g. the history of the steps performed to obtain the desired skim or the SQL query to select it).

It requires access to the desired tag database (authorization and drivers).

3.1.4. *Skim selector*. Extracts events references from a TAG database based upon metadata queries, and stores them in ROOT-based collections. Each collection is a list of tuples, containing event identifier - file GUID (Globally Unique Identifier); that identifies uniquely within ATLAS the events included in the skim. To identify uniquely the output of DSS, further information defining the slimming and thinning part of the selection is required. This is done using the Athena jobOption file that allows one to modify the selected events: the user can choose the full event, use options that keep only part of the event, or have maximum flexibility providing a custom jobOption.

It uses information identifying a skim selection, as the one provided by the skim composer (e.g., a tag database and the query identifying the selection). This information is accumulated and stored in the SkimDB.

It provides one or more lists identifying uniquely the events in the skim (i.e. "root" collection files and an associated PoolFileCatalog.xml containing the physical location of the files).

It requires access to the tag database (authorization and drivers), POOL utilities to manage ATLAS data, and the DDM system or a local catalog to locate the files. Globus tools (from VDT[17] or OSG[18]) are also required to access the DDM system.

3.1.5. *Skim extractor*. Extracts events from an AOD (or any other sample) using the TAG input provided by the skim selector. This consists of retrieving the necessary input datasets (e.g., using DQ2 and transfer tools), running Athena with the input and the jobOptions provided by skim selector and making the skimmed files/dataset available in a staging area. The skimming jobs can run: 'locally', in a local queue (Condor, PBS, ...), or on the Grid (also using a distributed analysis system, e.g., Pathena). Embedded in the skim extractor is also a splitting functionality. In fact it can either use a single collection file, or split a collection file creating a job for each input file (GUID) with selected events, or split collection files ensuring to each job at least a given minimum number of events.

It uses information about the extraction method, eventual slimming and thinning options and one or more lists identifying uniquely the events in the skim (i.e. "root" collection files and associated PoolFileCatalog.xml containing the physical location of the files).

It provides the skimmed dataset: one or more files containing the desired output events.

It requires an environment to execute the extraction:

- if the extraction is local, the ATLAS release;
- if jobs are submitted to a Computing Element (a queue or the Grid), its description;
- if the extraction use the distributed analysis system, it has to be installed and configured.

It further requires POOL utilities to manage ATLAS data and the DDM system or a local catalog to locate the files. Globus tools (from VDT or OSG) are also required to access the DDM system.

3.1.6. *Skim registrar*. Register the skimmed dataset into the DDM system: define the dataset and register it (most likely after copying its files to the local SE). A skimmed dataset should be considered complete by definition, even if the source dataset is open. The skim is a cut done at a well-defined point in time on some data. If the data changes we have a different skim. This step is not performed if the skimmed dataset is used only locally.

It uses the skimmed dataset (name, files and metadata).

It provides a registration of the skimmed dataset.

It requires the DDM system or a local catalog to locate the files. Globus tools (from VDT or OSG) are also required to access the DDM system.

3.1.7. *Skim publisher*. All information concerning a skim is recorded in the SkimDB. This component registers skims produced outside DSS. A browser of the SkimDB can retrieve past skims and a notification service may allow users to know when a skim is ready. A local cache for skimmed datasets would make even easier and more efficient the retrieval of past skims.

It uses registration data provided by the user (if an external skim is entered).

It provides access to past skims.

It requires a user interface for the SkimDB and may require a notification system (callbacks, Nagios, …) or disk space for caching results.

3.2. DSS Implementation: the prototype at MWT2

The current Data Skimming Service prototype, deployed at the ATLAS Midwest Tier2 (MWT2_UC), implements the above design using the Django[19] Web framework. The previous section described the model, here the focus are the code and additional elements like the user interface, integration with other software, the database backend and the software footprint and distribution.

Django developers define it as "a high-level Python Web framework that encourages rapid development and clean, pragmatic design.". A Web framework is software that eases the building dynamic Web sites: it provides a method of mapping requested URLs to code that handles requests; it makes it easy to display, validate and redisplay HTML forms; it converts user-submitted input into data structures that can be manipulated conveniently; it helps separate content from presentation via a template system and it conveniently integrates with storage layers. Django was chosen mainly because it is Python based and it eases the rendering of HTML pages thanks to its template mechanism and the mapping of the URLs to the code that they invoke. DSS is developed using a "model-view-controller" (MVC) architecture and the "don't repeat yourself" (DRY) principle. In a MVC architecture the code is loosely coupled, separated by a clear API. This is especially true for the code for defining and accessing data (the model), which is separate from the user interface (the view, rendering the HTML), which in turn is separate from the business logic (the controller, executing the ATLAS code). DRY software avoids repeating the same information in different places, e.g., using constants that are defined once, using shared configuration and data introspection.

The Web interface is the GUI that allows scientists to interact easily with the Data Skimming Service. It further integrates different technologies in a seamless system, masking complex and different interactions. Depending on their needs, users can choose any known TAG database for their skims and DSS supports transparently local or remote, Oracle or MySQL databases. The modularity and loose coupling of DSS allows also an easy integration of external components. For example the

CSC TAGDB Browser interface[11] is used as browser and selector for the skims of the CSC[20] streams. Its composer/selector offers more features than DSS's one and can be used to produce a ROOT collection with the event selection that is then imported in DSS to complete the skimming. Under evaluation is the integration of another browser: an applet based upon JAS3[21]. This would allow interactive analysis and the generation of plots or histograms of the selected events. The DSS prototype provides only skimming capabilities: in order to alter the events the user would have to replace the default Athena jobOption file used by DSS with a custom one containing the desired instructions.

The database backend, including the SkimDB, is currently a sqLite database but it will be easy to replace it with more powerful database servers like MySQL or Postgres whenever necessary to increase performance or scale-up.

The system has a small footprint by design. The Data Skimming Service (excluding the event extraction performed with Athena) requires only Python, a database (e.g., sqLite) and, if the execution is not local, the ability to submit jobs to the queue or issue Grid commands and DQ2 requests (Grid and DQ2 client tools). In order to increase reliability and reduce the dependency on external systems DSS can scale from running completely on a workstation to make full use of Grid resources. When running on a single computer it uses a local MySQL tag database, locally installed ATLAS releases for the skim extraction and local files as file catalogs (PoolFileCatalog.xml). If the tag database is not local it can connect to remote MySQL or Oracle databases. It can use a local computing element, a Condor or PBS cluster, the neighboring storage element and its Local Replica Catalog (LRC, a file catalog used in US-ATLAS) to run the jobs for the skim extraction. Extraction jobs can also be submitted to the Grid, and it is relatively easy also to integrate a distributed analysis system like Pathena[15] or GANGA[22] to perform the skim extraction.

The preferred environment for DSS is a computing site like a Tier2, with a local storage element containing most of the data possibly involved in skims, and with computing elements to execute the extraction of the skim. The addition of the service API will enable also neighboring Tier3s to take advantage of the skimming capabilities offered at the Tier2.

4. Processing workflows for Analysis and Managed Production in ATLAS

The Data Skimming Service is providing a convenient and efficient framework to perform skimming operations. The concept is not new: a scientist can skim a dataset running a filter on the events of the dataset.

The use of a relational database with tag information has been proven to be more performant than direct filtering of the events. The advantage is even larger for big datasets, for complex queries and especially for highly selective filters [11]. Therefore it is preferable to use DSS or a similar service interacting with a tag database at least to produce the collection with the selected events.

Furthermore, whenever possible, it is preferable to limit the execution of the extraction to the closest resource with all the necessary input files and sufficient computing power. This will involve fewer resources and be more reliable. If a job is short and can be run locally, the scientist will have more control, he will not rely on anything more than the local machine and will be closer to his data. If the data resides in a local storage element and there is a local Condor or PBS queue, DSS will be able to run the skim without requiring any external service. Other workflow managers used in ATLAS managed production and distributed analysis, like Panda[15], GANGA[22] and previously Capone[23] and other executors require central servers to collect or dispatch the jobs and increase the size of the interaction to the Grid. The skimming jobs could be sent to a different computing element even if there are resources locally and all the input files are in the local storage element. As stated in error statistics included in [15], after "ATLAS application error" and "Job killed by pilot" (errors that cannot be affected by changing the workflow management), the next three most common causes of errors are related to data transfer and together cover almost 30% of the errors. Considering the high replication of most AOD, DSS would probably allow running on the local resources, eliminating those possible causes of error.

The Tag Navigator Tool (TNT)[11] is the closest project to DSS within ATLAS: they both access TAG information to drive the skimming process, and a POOL utility developed by the TNT team is being used also by DSS. TNT is a more mature project, available in two flavors: as a standalone tool, performing data skimming on the grid LCG, and as a plugin for GANGA, distributed with GANGA itself as a Splitter that coordinates the skim extraction executed through GANGA. DSS, while still a prototype, provides a GUI while TNT has only a command line interface. The main difference between the two as standalone services is that while TNT extraction jobs run on LCG, DSS jobs run on OSG and the two grids are not easily interchangeable: ATLAS is using different file catalogs and a different execution environment. Job submission in GANGA can support different backends but at the moment has no good support for OSG. For DSS it may be interesting to use GANGA, to provide a skim extractor module supporting LCG and the other GANGA backends.

5. Conclusions and Acknowledgements

This paper presented the Data Skimming Service, which is a tool to facilitate the selection of desired events within big datasets. This tool is particularly convenient when the data to skim is present at the facility offering the service, e.g. operating within a Tier2. The current prototype requires robust user interaction; it could be simplified automating some tasks to make it more user friendly. Furthermore it would benefit from the possibility to use Panda and GANGA as backends for skim extraction. Anyway, considering that in the USA there are plans to host at each Tier2 a full replica of the AOD set, DSS could help greatly local scientists and the Tier3s that are leveraging the resources of neighboring Tier2s.

This research was supported in part by the U.S. Department of Energy. Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under contract DE-AC02-06CH11375

References

[1] The Large Headron Collider Project at CERN, http://lhc.web.cern.ch/lhc/
[2] CERN LHC Experiments:
A Toroidal LHC ApparatuS (ATLAS), http://atlas.web.cern.ch/Atlas/
Compact Muon Solenoid (CMS), http://cms.cern.ch/
The Large Hadron Collider beauty experiment (LHCb), http://lhcb.web.cern.ch/lhcb/
A Large Ion Collider Experiment (ALICE), http://aliceinfo.cern.ch/
[3] ATLAS Computing TDR, CERN-LHCC-2005-022 (2005).ß
[4] ATLAS Midwest Tier2, http://www.mwt2.org/
[5] Don Quijote2 (DQ2), the ATLAS Distributed Data Management (DDM), https://twiki.cern.ch/twiki/bin/view/Atlas/DistributedDataManagement
[6] D. Costanzo, et. al., "Metadata for ATLAS", CERN-ATL-COM-GEN-2007-001 (2007).
[7] POOL - Persistency Framework, http://lcgapp.cern.ch/project/persist
[8] ROOT, http://root.cern.ch
[9] A. Vaniachine, H. Von Der Schmitt "Development, Deployment and Operations of ATLAS Databases", CHEP2007 (2007)
[10] K. Assamagan, et. al., "Report of the Event Tag Review and Recommendation Group", ATL-COM-SOFT-2006-003 (2006).
[11] C. Nicholson, et. al., "Integration of the ATLAS Tag Database with Data Management and Analysis Components", presented at CHEP2007 (2007).
[12] P. Fuhrmann et al, "dCache, a distributed data storage caching system", CHEP2001 (2001)
[13] A. Dorigo, P. Elmer, F. Furano, A. Hanushewsky, "XROOTD - A highly scalable architecture for data access", WSEAS – Prague (2005), http://xrootd.slac.stanford.edu/
[14] Storage Resource Manager, http://sdm.lbl.gov/srm-wg/doc/v3/SRM.func.spec.v3.0.rc1.html
[15] P. Nilsson, "Experience from a Pilot based system for ATLAS", CHEP2007 (2007)
[16] The Globus Alliance, http://www.globus.org/

[17] The Virtual Data Toolkit (VDT), http://www.lsc-group.phys.umw.edu/vdt/
[18] Open Science Grid, http://www.opensciencegrid.org/
[19] The Django Project, http://www.djangoproject.org/
[20] Computing System Commissioning, https://twiki.cern.ch/twiki/bin/view/Atlas/CSCNotesList
[21] V. Serbo et al. "JAIDA, JAS3, WIRED4 and the AIDA tag library - experience and new developments", CHEP2007 (2007)
[22] A. Soroko et al. "The GANGA user interface for physics analysis on distributed resources", CHEP2004 (2004)
[23] M. Mambelli, R. Gardner, et al. "ATLAS Data Challenge Production on Grid3", Proc. of CHEP04, Interlaken, CH, Sept 2004