ATLAS Grid Information System for the ATLAS Experiment

Raquel Pezoa[1,3], Simone Campana[4], Benjamin Gaidioz[4],
Gilbert Poulard[4], Ricardo Rocha[4], Luis Salinas[1,2]

[1] *Center for Technological Innovation on High Performance Computing, Universidad Técnica Federico Santa María, Valparaíso, Chile*
{raquel.pezoa, luis.salinas}@usm.cl

[2] *Informatics Department, Universidad Técnica Federico Santa María Valparaíso, Chile; luis.salinas@usm.cl*

[3] *Physics Department, Universidad Técnica Federico Santa María Valparaíso, Chile; raquel.pezoa@usm.cl*

[4] *European Organization for Nuclear Research, CERN, Geneva, Switzerland*
{benjamin.gaidioz, gibert.poulard, ricardo.rocha}@cern.ch

Abstract

ATLAS is the largest high energy physics experiment ever undertaken in the history of Science and it will produce huge volumes of data, of the order of 10 PB per year. ATLAS uses grid technology to distribute, store and analyse these immense amounts of data. The ATLAS Distributed Computing (ADC) system provides a set of tools and libraries enabling data movement, processing and analysis on a grid environment. While reaching a state of maturity high enough for real data taking, it became clear that one component was missing exposing consistent information regarding site topology, service and resource information from all three distinct ATLAS grids (EGEE, NDGF and OSG).

This paper describes the ATLAS Grid Information System (AGIS) attempts to overcome that lack, retrieving, storing and deploying the static and semi-static information about resources, services, and topology of the whole ATLAS grid. We report on the fundamental architectural work and the basic implementation of the ATLAS Grid Information System.

1. CERN and the LHC

CERN, the **European Organization for Nuclear Research** is one of the world's largest and most renowned centres for scientific research in high energy physics.

The Large Hadron Collider (LHC) [1] is a gigantic scientific instrument located at CERN near Geneva, Switzerland. The LHC has the form of a large ring of 27 Km length and lies some 100 meters underground, below the Franco-Swiss border to the west of Geneva, at the foot of the Jura mountains, in front of the Alps. The LHC is a particle accelerator which brings protons and ions into head-on collisions at energies much higher than ever achieved

before. The LHC hosts four experiments in High Energy Physics: ALICE, ATLAS, CMS and LHCb. Each experiment has its own system of detectors. These detectors will produce huge amounts of data related to the myriads of particles --known and unknown-- resulting from the collisions. This immense volume of data must be handled by the computing and information systems integrated into the LHC. One of the experiments to be carried on at the LHC is called ATLAS, which is an acronym that stands for **A** **T**oroidal **L**HC **A**pparatu**S**.

ATLAS is a multi-grid environment, that is, it is composed by diverse grid infrastructures (called generically sub-grids in the sequel). For a multi-grid system to work efficiently it is of paramount importance to have a system that provides information regarding resources, services and topology of the grid. The ATLAS sub-grids currently have systems that solve this issue for each particular sub-grid, but there is a real need of a global information system operating above the particular sub-grids. This paper reports about a solution to the above problem by means of a system called **A**TLAS **G**rid **I**nformation **S**ystem (AGIS), which is operating in a global way above the local sub-grid information systems. The purpose of this system is to store and to provide static and semi-static information about resources, services and topology of the whole ATLAS grid. This information is retrieved from different systems belonging to the different sub-grids. To accomplish these tasks, different collecting services were implemented. These collectors periodically gather the relevant data and store it into the local database of the ATLAS Grid Information System. To provide the information to the different components of the ATLAS grid, APIs were developed to query and update the system. Also, the data is available through a web application, which uses authentication service (X.509 proxy certificates) to update the system.

2. ATLAS Distributed Computing

The search for new fundamental particles in the ATLAS experiment is expected to produce about 10 Petabytes ($10x10^{15}$ Bytes) of data per year. In order to handle such a huge amount of data, both during online data acquisition and during off-line processing of the events, a distributed computing model called *ATLAS computing model* [3] was developed by the collaboration and has decisively contributed to the development of *hierarchical computing* and of the current *computer grid paradigm*, characterized by a high degree of decentralization and sharing of resources [4].

The ATLAS computing model is based on a worldwide computing grid infrastructure that uses a set of hierarchical tiers. The root node of the complex tiered topology is **Tier-0**, located at CERN itself. Around Tier-0 there are 10 big **Tier-1**, located in Europe, Asia and North America, each serving smaller communities usually tied to a specific Tier-1 by geographical proximity. The next level is **Tier-2** gathering more than 100 sites around the world. This number will probably increase when the next level, **Tier-3**, will start to grow gathering smaller centres around the world (see Figure 1).

The sites belonging to these tiers are also grouped into so called *clouds*. Thus, a rather complex network emerges, which is usually referred to as the ATLAS grid. As of today the ATLAS grid has more than 170 sites providing more than 30000 CPU's and a storage capacity of more than 11000 TB [5].

Furthermore, the ATLAS grid is composed by three independent and different sub-grids: **E**nabling **G**rid for **E**-scienc**E** (EGEE) [11], **N**ordic **D**ata **G**rid **F**acility (NDGF) [12] and **O**pen **S**cience **G**rid (OSG) [13]. These three sub-grids form the Worldwide LHC Grid Computing Grid infrastructure (WLCG). Each sub-grid contains its own middleware and hence, it own implementation of the different services of a grid system. However, the ATLAS grid has to work as a single coherent infrastructure. Thus, a common middleware layer is required. The

design and implementation of a new component in that layer – the ATLAS Grid Information System – is described in this paper.

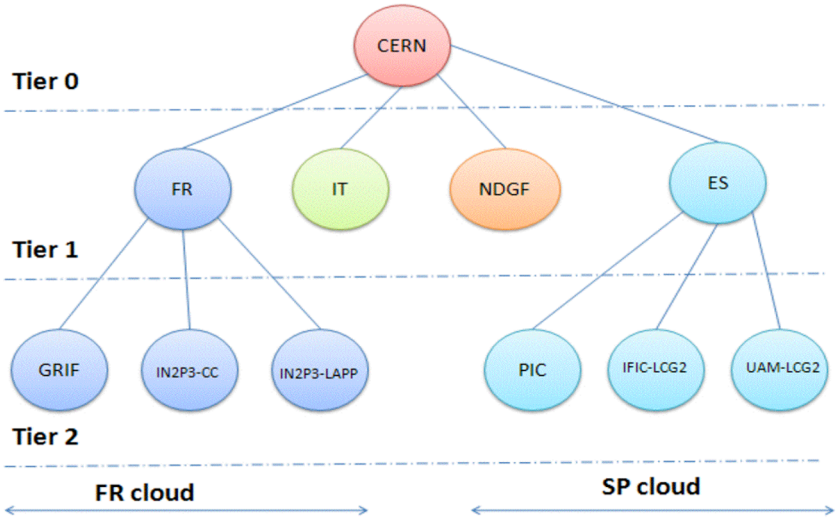


Figure 1. Tiers of ATLAS

3. ATLAS Grid Information System

As it was mentioned above, each sub-grid of the ATLAS grid contains its own middleware and hence its own implementation of the different services of a grid system. The ATLAS grid depends on the underlining information provided by the grid middleware. In order to optimally combine it with the monitoring information and the internal configuration of the ATLAS Distributed Computing layer, ATLAS developed a new component, the ATLAS Grid Information System (AGIS).

The main goal of the ATLAS Grid Information System (AGIS) is to provide the missing information regarding resources, services and topology of the whole ATLAS grid, in order to provide interoperability in a multi-grid environment. AGIS retrieves static and semi-static information from the diverse sources coming from the different sub-grids. AGIS was designed considering the benefits of the **Dashboard Framework** [6] which provides directives to model a system with a layered client-server architecture, following a *Model View Controlle*r (MVC) [7] pattern. Furthermore, AGIS follows the **RE**presentational **S**tate **T**ransfer (REST) principles [8], which are a set of architectural constraints on top of the basic client-server architecture style.

AGIS provides diverse services to the clients for data retrieval. These services are software components provided through a network-accessible endpoint. The AGIS architecture (Figure 2) groups the components in three main layers: *Data Access Layer*, *Collectors*, and *Client Connection*, which are in charge of data management, data retrieval and data exposure, respectively.

The MVC pattern isolates business logic from user interface, resulting in an application easy to modify either the visualization of the application or the underlying layers.

REST is a design principle for ultra large scale systems [9]. This principle allows the achievement of important design attributes such as loose-coupling, reliability, data visibility and interoperability.

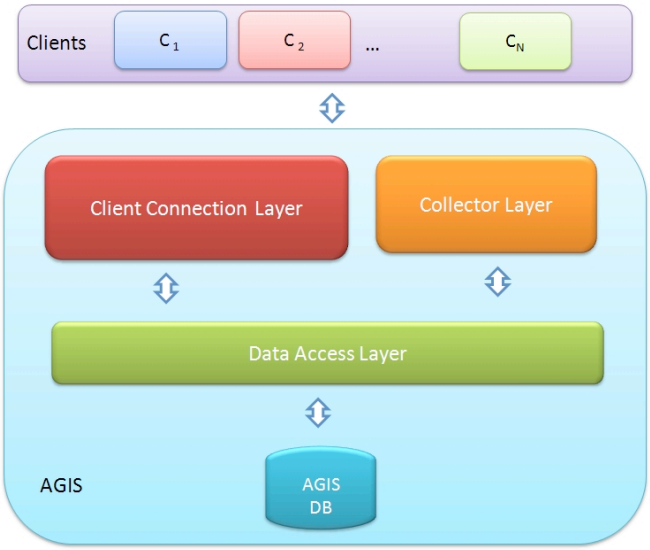


Figure 2. AGIS Architecture Overview

3.1. AGIS Components

The three layers (shown in Figure 2) logically group the main functionalities of AGIS: data collection, management of persistent data collected from the diverse sources, and the data exposing available in AGIS.

The *Data Access layer* corresponds to the data access interface and allows the management of the persistent data, which is stored in a RDBMS system. The components of this layer provide functionalities to query and update the stored data. Access to the database is accomplished by using a connection pool in order to increase the performance of the system.

The *Collectors layer* provides capabilities for the data collection. AGIS has six different collectors each one retrieving data from different sources. These collectors perform the service of data retrieving and processing, giving the proper format to the data, in order to be passed to the persistent storage (Data Access layer).

The *Client Connection layer* allows heterogeneous "application-to-application" and "user-application" communication. The clients may contact the system to retrieve AGIS data using direct HTTP access, APIs, command lines tools (CLI tools), or the web interface (for users).

The interaction between the users and applications with AGIS is done using HTTP requests. Thus, HTTP native features were used as much as possible in all parts of the implementation of the system.

AGIS follows the RESTful style. Therefore, resources in the RESTful scope are identified with an ID. These resources may be: a request for cloud information, a request for site that belongs to a determined tier, for the space tokens of a determined Storage Element service, etc. AGIS exhibits all interesting information as URL identifiable resources. The methods for manipulating these resources are standardized HTTP verbs, which increase process visibility and interoperability.

Therefore, when a client needs to access AGIS data, it must contact the AGIS services, which are accessed as AGIS resources identified with an URL (as the first principle of RESTful proposes). The AGIS resources are accessed using the standard methods (second principle of RESTful): PUT, GET, POST and DELETE and they are decoupled from their

representation (third principle of RESTful). Furthermore, every interaction with a resource is stateless and context-free, which promotes the scalability through the feasibility of cashing and reduced workload on the server. For instance, the AGIS cloud resource is referenced with an URL. A POST request to this resource results in the creation of a new cloud in the AGIS system. A GET request to the cloud resource will return a list of existing clouds; in addition, the client can add some parameters in the GET request in order to retrieve specific resource data. The DELETE request eliminates a cloud from the system.

3.2. Implementation

The ATLAS Grid Information System was implemented using the functionalities provided by the **Dashboard Framework**. Python is the programming language used to implement the system. The implementation had different stages, the development of Data Access Object (DAO), the APIs, the data collectors, the command lines (CLI) tools and the web interface, which are described as follows.

Data Access Object

The implementation of a *Data Access Object* (DAO) allows a clear design and maintainability of the system, because the application queries are decoupled from the internal implementation of the data storage. The DAO represents the data access interface, which are a public set of methods for the update and retrieval of information. The connection pool is the method used to access the database, in order to reduce the overhead in creating new connections, reducing the load on the server and increasing the performance.

APIs

AGISQuery and *AGISUpdate* are the two APIs developed to query and update, respectively, the ATLAS Grid Information System. The AGISQuery API provides different methods to query the ATLAS Grid Information System. The AGISUpdate API allows updating the data of the ATLAS Grid Information System. To update the system it is necessary to have some permission. The authentication is done via X509 proxy certificates.

Data Collectors

The data collection is done using services that retrieves periodically data from the different sources of information. These collector services are daemons that can contact diverse hosts and collect information of interest. The implementation is based on the *arda.dashboard.service-config* module provided by the Dashboard Framework.

There are six different sources of information, thus six different services that retrieve the data and store it into a local database, and these services are: *TiersOfATLASCollector*, *GOCDBCollector*, *OIMCollector*, *BDIICollector*, *PandaCollector*, and *NDGFCollector*.

The data collection implies regular access to the information sources. To provide a reliable ATLAS Grid Information System, the collector services should run constantly and need to recover any missing data in case of eventual failure and following restart. The functionalities of the Dashboard Framework allow to monitor the collectors and to be aware about the status of them, using some reporting methods: a web application showing the status of the collectors, an email or a SMS.

Command Line Tools

The command line (CLI) tools allow the users to update and retrieve information to or from the ATLAS Grid Information System, using command lines. The set of available tools and their functionality is presented in Table 1.

CLI Tool Name	Functionality
agis-clouds	Lists ATLAS clouds agis-clouds [-P | --pair]
agis-sites	Lists ATLAS sites agis-sites [-C | --cloud] [-T | --tier] [-R | --region] [-G | --grid] [-O | --country] [-N | --name]
agis-services	Lists ATLAS services agis-services [-S | --site] [-T | --type] [-E | --endpoint]
agis-spaceTokens	Lists ATLAS space tokens agis-spaceTokens [-C | --cloud] [-S | --site] [-E | --se]
agis-queues	Lists ATLAS queues agis-queues [-C | --cloud] [-S | --site] [-E | --ce]

Table 1. CLI Tools

Web Interface

This first version of a web interface allows visualizing and updating the AGIS data in an easy way. The web interface provides web pages, which are structured in the main components: topology, clouds, sites, and services.

As it was mentioned before, all the processes of adding, removing or updating information are done with authentication, using X509 proxy certificates. These certificates are commonly used in security systems are a standard for dynamic delegation and identity creation in public key infrastructures.

3.3. Evaluation

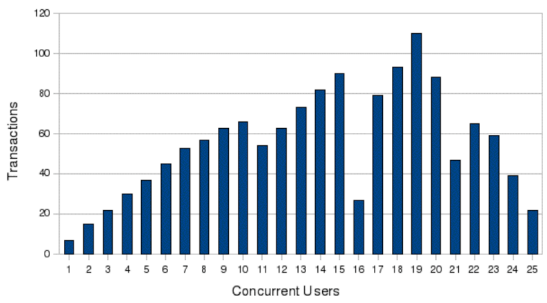
A first stage in the evaluation of AGIS is to determine if the functional requirements have been satisfied. Unit and integration tests were used to evaluate the functional requirements. In addition, it is important to know the functionalities of the system are performed. Thus, it is necessary to analyse the non-functional requirements, which are related with the *performance* of the system. In the AGIS system its performance is mainly related with the *scalability*.

Scalability is a term that appears frequently in computing literature, but is poorly defined and there is little consensus as to what the term actually means. However, according [10] the different definitions of scalability have a common aspect, which is a system needs to tolerate variation or scaling in some characteristic affecting its execution. AGIS scalability was evaluated considering the *number of concurrent clients* that query the system in a fixed period of time. The test methodology is illustrated in the code below.

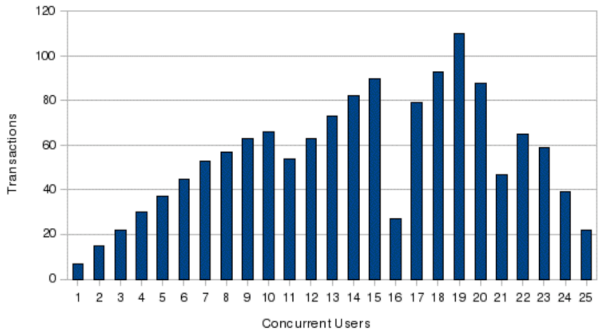
```
SET number of concurrent Users to N
SET period of time 10
SET number of maxConcUsers to P
SET AGISservices to AGISserv
WHILE concurrent Users <= maxConcUsers :
   FOR S in AGIS services:
         queryAGIS ( S , concurrentUsers , T)
         concurrentUsers ++
         PRINT numberOfTransactions
   END FOR
```

The tests increase the number of concurrent users (concurrentUsers) that are requesting a specific service (S) until the maximum of concurrent users (maxConcUsers) is reached, in a period of time (T). This test gives as output the number of transactions that can handle the system. Each requested service corresponds to a use case.

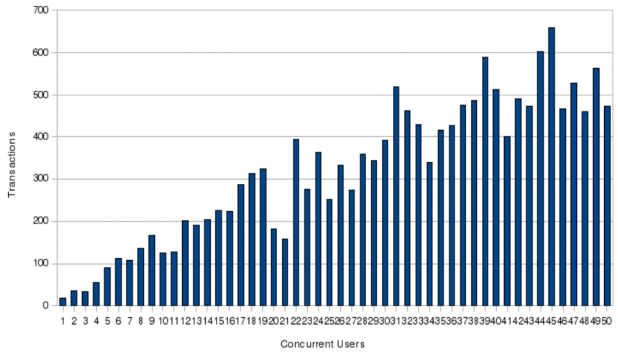
Three different test cases were developed. One of the test cases is called “GetCloud”, which returns the clouds of the ATLAS grid; it represents the simplest service of the system with minimum operations involved. The other test case is called “GetSite”, which returns all the sites of the system. Finally, “QueryTopology” queries all the entities and their relationships (providing the whole ATLAS grid structure), dealing with relative complicated computations.



a) GetSite



b) QueryTopology



c) GetCloud

Figure 3. AGIS Tests

Figure 3 shows some of the tests results, which corresponds to the number of transactions that AGIS can handle in a period of 10 seconds. GetSite and QueryTopology showed that AGIS was able to handle with a good performance up to 25 concurrent users. GetSite test had a peak of 19 concurrent users, which corresponds to 11.02 [trans/sec] (transactions per seconds) and QueryTopology had a peak of 15 concurrent users with 9.04 [trans/sec]. GetCloud had a better performance, with 45 concurrent users, obtaining 66.1 [trans/sec]. Thus, the results of the three use cases can be considered as "good results" due to the important is to satisfy the demands of the AGIS users, which can easily attain some hundreds in number. Therefore, 11.02 [trans/sec], [trans/sec] and 66.1 [trans/sec] to get information

regarding cloud, site and topology, respectively, can reflect that with the current AGIS implementation and server configuration, AGIS will be able to serve the expected load coming from the diverse services and end users of ATLAS.

4. Conclusions

The main purpose of this paper is to briefly report on the fundamental architectural work and the basic implementation of the ATLAS Grid Information System. For a detailed description of this system please refer to [14]. The ATLAS grid corresponds to a multi-grid environment because it is composed by three different sub-grids: EGEE, NDGF and OSG. Due to the multi-grid nature of ATLAS, various components have been designed and implemented that operate above the middleware components of each sub-grid in order to provide interoperability. However, there was a need of a component that provides the functionalities of a global Grid Information System. The ATLAS Grid Information System (AGIS) is the new component that was designed and developed in order to overcome that lack. AGIS retrieves and provides the static and semi-static information related with the resources, services and topology of the ATLAS grid. To accomplish this task six different collecting services had to be implemented. These collectors must periodically collect the relevant data and store it into the local database of AGIS. To provide the information to the different components of the ATLAS grid, APIs were developed to query and update the system. Also, the data is available through a web application, which uses authentication service to update the system.

The design of AGIS used the REST (**RE**presentational **S**tate **T**ransfer) style as the approach to provide interoperability, data visibility and scalability. In the computer science literature no other grid information system using this style was found. The REST choice gave good results. Some tests were applied to AGIS, aiming to evaluate the performance of the system, in order to check that the design functional and non-functional system requirements were indeed satisfied. Considering that the AGIS users easily represent hundreds together and that the tests were applied in the real AGIS that is currently working in the ATLAS grid, we can safely say that AGIS was able to serve the expected load and the functional and non-functional system design requirements. AGIS handled a peak of 661 [trans/sec] having 46 concurrent users in a period of 10 seconds, which is enough for a system that provides static and semi-static information regarding services, resource and topology in the ATLAS grid.

5. Future Work

The system is still under development, and we are trying to include requirements from more areas than the ones initially considered. Furthermore, it is necessary to improve some components --such as the web interface of the system-- and also to continue with the evaluation phase.
The system will be extended to include service metadata, mostly central configuration data for various components in the system.

6. Acknowledgements

Raquel Pezoa deeply appreciates the financial support received from two HELEN fellowships that allowed her to complete two long-term and very pleasant and productive stays (April-October, 2007, and May-July, 2008) at CERN, Geneva, Switzerland, in order to develop researches on grid computing.

References

[1] CERN. Large Hadron Collider. http://public.web.cern.ch/Public/en/LHC/LHC-en.html

[2] ATLAS Experiment. http://atlas.ch/

[3] The ATLAS Computing Model.
http://www.gridpp.ac.uk/eb/ComputingModels/atlas_computing_model.pdf

[4] ATLAS Distributed Computing.
https://twiki.cern.ch/twiki/bin/view/Atlas/AtlasDistributedComputing

[5] Gilbert Poulard, ATLAS and the WLCG Project. Latin American Software and Computing Workshop. 10 - 13 March 2008, Buenos Aires, Argentina.
http://www.df.uba.ar/~aia/atlas_workshop/index.html

[6] Julia Andreeva et all. Dashboard for the LHC Experiments. In Journal of Physics: Conference Series. Institute of Physics Publishing, 2007.

[7] Avraham Leff and James T. Rayfield. Web-application Development Using the Model/View/Controller Design Pattern. In EDOC '01: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing, page 118, Washington, DC, USA, 2001. IEEE Computer Society.

[8] Roy T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine, 2000.

[9] Xiwei Xu, Liming Zhu, Yan Liu, and Mark Staples. Resource-oriented Business Process Modeling for Ultra-Large-Scale Systems. In ULSSIS '08: Proceedings of the 2nd international workshop on Ultra-Large-Scale Software-Intensive Systems, pages 65-68, New York, NY, USA, 2008. ACM.

[10] Leticia Duboc, David Rosenblum, and Tony Wicks. A Framework for Characterization and Analysis of Software System Scalability. In ESEC-FSE '07: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pages 375-384, New York, NY, USA, 2007. ACM.

[11] Enabling Grid for E-sciencE. http://www.eu-egee.org/

[12] Nordic Data Grid Facility. http://www.ndgf.org

[13] Open Science Grid. http://www.opensciencegrid.org/

[14]Raquel Pezoa Rivera, Design and Implementation of a Grid Information System for the ATLAS Experiment at CERN. Thesis. In partial fulfillment of the requirements for the degree of Master in Sciences in Computer Science. Universidad Técnica Federico Santa María, Valparaíso, Chile, November 2008.