The Advanced Resource Connector for Distributed LHC Computing

David Cameron[*ab], **Aleksandr Konstantinov**[a], **Farid Ould-Saada**[a], **Katarina Pajchel**[a], **Alexander Read**[a], **Bjørn Samset**[a], **Adrian Taga**[a] [†]

[a]*Department of Physics, University of Oslo, P.b. 1048 Blindern, N-0316 Oslo, Norway*
[b]*Nordic Data Grid Facility, Kastruplundgade 22, DK-2770 Kastrup, Denmark*
E-mail: cameron@ndgf.org

The NorduGrid collaboration and its middleware product, ARC (the Advanced Resource Connector), span institutions in Scandinavia and several other countries in Europe and the rest of the world. The innovative nature of the ARC design and flexible, lightweight distribution make it an ideal choice to connect heterogeneous distributed resources for use by HEP and non-HEP applications alike. ARC has been used by scientific projects for many years and through experience it has been hardened and refined to a reliable, efficient software product. In this paper we present the architecture and design of ARC and show how ARC's simplicity eases application integration and facilitates taking advantage of distributed resources. Example applications are shown along with some results from one particular application, simulation production and analysis for the ATLAS experiment, as an illustration of ARC's common usage today. These results demonstrate ARC's ability to handle significant fractions of the computing needs of the LHC experiments today and well into the future.

XII Advanced Computing and Analysis Techniques in Physics Research
November 3-7, 2008
Erice, Italy

*Speaker.
†This work was partly supported by the Nordic Data Grid Facility and EU KnowARC (contract nr. 032691).

1. Introduction

NorduGrid [8] is a Grid Research and Development collaboration aiming at development, maintenance and support of the free Grid middleware, known as the Advance Resource Connector (ARC) [4]. Institutes and projects from several European countries participate in NorduGrid and contribute to ARC. ARC was first conceived in 2001, to meet the urgent needs of users in the Nordic countries wishing to harness Grid computing resources. Existing projects such as the Globus Toolkit [2], UNICORE [1] and the European DataGrid [3] were either missing required features or were scheduled to finish too far in the future to be useful. Hence the NorduGrid collaboration started to develop its own Grid middleware solution, built on top of components of the Globus Toolkit. By 2002, ARC was already in use in a production environment managing thousands of jobs performing Monte-Carlo simulation for the ATLAS [5] experiment across several sites in several countries. These jobs have been running almost non-stop at an increasing scale ever since then, along with jobs from other scientific communities, in parallel to the rapid growth in the amount of computing resources managed by ARC. At the end of 2008, ARC was managing over 25000 CPUs in more than 50 sites across 13 countries [9].

2. ARC Architecture and Design

An important requirement from the very start of ARC development was to keep the middleware portable, compact and manageable both on the server and the client side. The basic set of high-level requirements is that the system must be efficient, reliable, scalable, secure and simple.

Perhaps the most important requirement that affects ARC architecture stems from the fact that most user tasks deal with massive data processing. A typical job expects that a certain set of input data is available via a POSIX open call. Moreover, such a job produces another set of data as a result of its activity. These input and output data might well reach several Gigabytes in size. The most challenging task of the Grid is thus not just to find free cycles for a job, but to ensure that the necessary input is staged in for it, and all the output is properly staged out. The only adequate approach is to include massive data staging and disk space management into the computing resource front-end functionality. The decision to include job data management functionality into the Grid front-end is one of the cornerstones of the architecture, adding to high reliability of the middleware and optimizing it for massive data processing tasks.

Another important requirement is that of a simple client that can be easily installed at any location by any user. Such a client must provide all the functionality necessary to work in the Grid environment and yet remain small in size and simple in installation. The functionality includes Grid job submission, monitoring and manipulation, the possibility to declare and submit batches of jobs with minimal overhead, and means to work with the data located on the Grid. The ARC client fulfils these requirements, exposing the functionality through a Command Line Interface, and C++ and Python Application Programming Interfaces (see Section 3).

In ARC, a major effort was put into making the resulting Grid system stable by design by avoiding centralised services, for example for job brokering. There are three main components in the architecture:

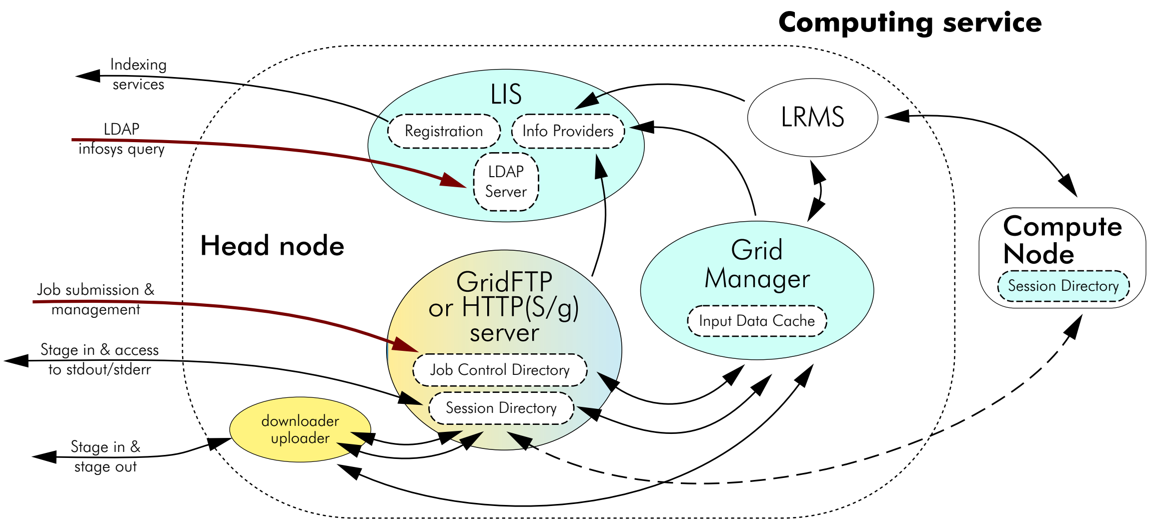

Figure 1: ARC server architecture, showing the set of services which enable a Grid computing resource.

1. The *Computing Service* (see Figure 1), which is implemented as a GridFTP-Grid Manager pair of services. The Grid Manager is the "heart" of the Grid, instantiated at each computing resource's (usually a cluster of computers) front-end as a new service. It serves as a gateway to the computing resource through a GridFTP channel. The Grid Manager provides an interface to the Local Resource Management System (LRMS), facilitates job manipulation, data management (download and upload of input and output data, and local caching of popular data), and allows for accounting and other essential functions.

2. The *Information System* serves as the "nervous system" of the Grid. It is implemented as a hierarchical distributed database: the information is produced and stored locally for each service (computing, storage) in the Local Information System (LIS, contained within the Computing Service), while the hierarchically connected *Index Services* maintain the list of known resources.

3. The *Brokering Client* is the "brain" of the Grid and the Grid user's interface. It is enabled with powerful resource discovery and brokering capabilities to distribute the workload across the Grid. It also provides client functionality for all the Grid services, and yet is lightweight and installable by any user in an arbitrary location in a matter of few minutes.

This combination of components gives two distinct advantages over other Grid middleware designs. Firstly, no Grid data management is performed on the computing nodes, and so access to storage and data indexing services is controlled rather than chaotic. Secondly, there is no resource broker service between the client and the resources - all resource brokering is done by each user's client.

3. Application Integration with ARC

The major user of ARC is currently the ATLAS experiment, performing Monte-Carlo generation and reconstruction of simulated data in order to prepare and test both the ATLAS software and the Grid infrastructure for the real data which is due to arrive in 2009. Physics tasks consisting of up to tens of thousands of Monte Carlo simulation or reconstruction jobs are defined by physics teams and assigned to one of the 3 Grids used by ATLAS roughly according to their capacities (the Nordic countries have agreed to provide 5.6% of the required computing and storage resources for the ATLAS experiment, but have consistently provided more than 10% over the last 2 years).

These jobs typically run from 30 minutes to 24 hours depending on the type of simulation. All the interactions with jobs on NorduGrid are handled by an "executor" which performs a set of call-outs to prepare jobs, to check their status, and to post-process the results. The executor is written in Python and interacts via the Python API of the NorduGrid ARC middleware. Figure 2 shows the basic Python code that is used to interact with ARC.

```
from arclib import *  # import ARC API

clusters = GetClusterResources() # get list of clusters from infosys
queuelist = GetQueueInfo(clusters, ...) # get queues on each cluster

xrsl = '&(executable="/bin/sh")...' # construct job description
X = Xrsl (xrsl)
targetlist = ConstructTargets (queuelist, X) # job submission targets
targetlist = PerformStandardBrokering (targetlist) # rank the targets

submitter = JobSubmission (X, targetlist) # job submitter object
jobid = submitter.Submit() # submit job

jobinfos =  GetJobInfo(jobids, ...) # check status
```

Figure 2: ATLAS use of ARC Python API.

Firstly the information system is queried for a list of possible job submission targets. Then the job description is constructed in the eXtended Resource Specification Language (XRSL), an extended version of the Globus RSL [7]. Brokering is performed by matching the job description against the possible targets and producing a ranked list. The job is then submitted, and later the status can be polled. The ease of use of the ARC API makes it suitable for any application wishing to take advantage of Grid resources.

4. Results

Using ARC has proven very successful for ATLAS. In 2008 just over one million jobs ran on ARC resources, around 10% of the total ATLAS jobs run worldwide, as shown on the ATLAS Dashboard [6]. The number of jobs run per day from mid-March to mid-December 2008 can be seen in Figure 3. The green bars represent successfully completed jobs and red bars represent jobs which failed.

The efficiency of these jobs (ratio of successfully completed jobs to total submitted jobs) was 92.4%, and the walltime efficiency (ratio of CPU time used for successful jobs to total CPU time used) was 97.1%. Both these values are far higher than either of the other Grids used by ATLAS. This can be attributed to two main factors - that data transfer does not use CPU cycles and

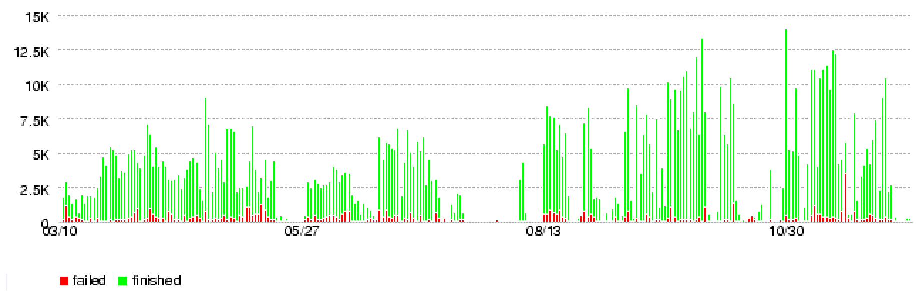

Figure 3: ATLAS ARC jobs completed per day for 9 months in 2008.

hence data transfer problems do not contribute to walltime loss, and that the close-knit and highly experienced NorduGrid community is fast and effective at reacting to any problems that do occur.

5. Conclusion

This paper has presented the ARC middleware and shown how simple it is to use distributed resources to obtain scientific results. Although initially designed for high-energy physics applications, ARC is now being used by scientific projects in a variety of diverse fields, from medical imaging to bioinformatics. ARC itself continues to evolve, and a large amount of effort is now focussed on enabling interoperability between ARC and other Grids through standards-based web services. However, the simple and lightweight design will remain, and continue to be attractive to an ever-expanding number of scientific communities.

References

[1] D. Erwin, editor. *Unicore Plus Final Report - Uniform Interface to Computing Resources*. Unicore Forum, Forschungszentrum Julich, 2003.

[2] I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[3] E. Laure et al. *The EU DataGrid Setting the Basis for Production Grids*. *Journal of Grid Computing*, 2(4):299–400, 2004.

[4] M Ellert *et al.* *Advanced Resource Connector middleware for lightweight computational Grids*. *Future Generation Computer Systems*, 23:219–240, 2007.

[5] The ATLAS Collaboration. ATLAS - A Toroidal LHC ApparatuS. Web site. URL http://atlas.web.cern.ch.

[6] The ATLAS Dashboard. Web site. URL http://dashb-atlas-prodsys.cern.ch/dashboard/request.py/summary.

[7] The Globus Alliance. The Globus Resource Specification Language RSL v1.0. Web site. URL http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html.

[8] The NorduGrid Collaboration. Web site. URL http://www.nordugrid.org.

[9] The NorduGrid Monitor. Web site. URL http://www.nordugrid.org/monitor/.