Adaptive data management in the ARC Grid middleware

D Cameron\(^1,2\), A Gholami\(^2,3\), D Karpenko\(^1\) and A Konstantinov\(^1,4\)

\(^1\) Department of Physics, University of Oslo, P.b. 1048 Blindern, N-0316 Oslo, Norway
\(^2\) Nordic Data Grid Facility, Kastruplundgaard 22, DK-2770 Kastrup, Denmark
\(^3\) PDC Center for High Performance Computing, The Royal Institute of Technology, 100 44 Stockholm, Sweden
\(^4\) Vilnius University Institute of Applied Research, Saultekio al. 9 III r., 10222 Vilnius, Lithuania

E-mail: cameron@ndgf.org

Abstract. The Advanced Resource Connector (ARC) Grid middleware was designed almost 10 years ago, and has proven to be an attractive distributed computing solution and successful in adapting to new data management and storage technologies. However, with an ever-increasing user base and scale of resources to manage, along with the introduction of more advanced data transfer protocols, some limitations in the current architecture have become apparent. The simple first-in first-out approach to data transfer leads to bottlenecks in the system, as does the built-in assumption that all data is immediately available from remote data storage. We present an entirely new data management architecture for ARC which aims to alleviate these problems, by introducing a three-layer structure. The top layer accepts incoming requests for data transfer and directs them to the middle layer, which schedules individual transfers and negotiates with various intermediate catalog and storage systems until the physical file is ready to be transferred. The lower layer performs all operations which use large amounts of bandwidth, i.e. the physical data transfer. Using such a layered structure allows more efficient use of the available bandwidth as well as enabling late-binding of jobs to data transfer slots based on a priority system. Here we describe in full detail the design and implementation of the new system.

1. Introduction

In a decentralised Grid environment there are several types of data access patterns. At one extreme there are solutions trying to provide POSIX-like access to any data across the Grid. At the other extreme the burden of accessing data is put on to the computational job itself by middleware.

Historically, the Advanced Resource Connector (ARC) middleware \([1]\) lies in the middle, providing POSIX local access to data specified in the job description. This approach is similar to what is implemented by many cluster batch systems but ARC also handles pre-staging and post-staging of specified data files to and from the location where the job is executed. The disadvantage of such a method is that the job itself needs to handle any data which it discovers it needs during run-time. ARC also deals with Grid-specific features, including a wide range of data access protocols, data caching, sophisticated access controls, data availability information, etc.
ARC currently uses a simplistic first-in-first-out approach to data management which does not involve any data transmission scheduling or prioritising. Data requirements for jobs are processed in the order they are received. But with larger clusters adopting ARC and a growing range of data access patterns it is becoming clear that this approach cannot scale indefinitely. Therefore a new architecture is required, which aims at providing optimised data staging capabilities, more effective use of supported protocols and better error handling, all hosted by a flexible and extensible framework.

This paper describes this architecture and its implementation, and is structured as follows: in Section 2 we describe current ARC use cases and the problems arising from those use cases. Section 3 discusses related work and existing solutions to these types of limitations. We propose an architecture for our own solution in Section 4 and the detailed implementation of this framework is presented in Section 5. We conclude in Section 6 by describing the current status of our implementation and future work.

2. Description of the problem and requirements
The principal ARC workload comes from analysis, Monte-Carlo simulation and re-processing of data from the ATLAS high-energy physics experiment [2] at CERN. Several Petabytes (10^15 bytes) of data are produced by the detector per year and distributed across a world-wide Grid for analysis [3]. In addition, simulation of the detector is done in order to model its expected performance and output, and twice-yearly the entire raw detector data is reconstructed as software and understanding of the detector improves.

All of this places large data management demands on the Grid system, and whilst simulation and reconstruction activities are somewhat predictable and controlled, analysis tends to be more chaotic in nature. Analysis also tends to involve running jobs which use a large amount of input data and run for a relatively short time, which leads to data management becoming the bottleneck in the system. The chaotic nature of analysis can be seen in the results of recent analysis of ATLAS data popularity [4]. They show that, far from evenly-spread popularity as predicted by the ATLAS computing model, a small fraction of data is accessed extremely often while a large part is accessed rarely or never. This also suggests that ARC’s ability to cache popular data should benefit ATLAS data analysis.

It is this analysis activity, which has increased tremendously since the Large Hadron Collider started full-scale operation in April 2010, along with experience from other ARC users, that has revealed some limitations in ARC. The following list outlines the most important issues and gives some ideas for how to solve them:

- Data transfers are done per job. As a result any problem with at least one of the files required by this job causes other jobs to be blocked from execution. It is therefore better to use file-level granularity when scheduling data transfer.
- Special features of sophisticated protocols are not taken into account. An example is SRM [5] with its support for tape sources and asynchronous negotiation of physical transfer URLs. ARC waits for the negotiation to complete or the file to be staged to disk and this can block other transfers and jobs. To overcome this problem, transfer negotiation can be made independent of physical data transfer, and therefore data transfers which have to wait for several hours after the initial request before the physical file is ready to be transferred can be efficiently supported.
- There is no support for priorities. A priorities-based approach is especially important inside big Virtual Organisations (VO) [6] with sophisticated internal structure. The new architecture should at least give the ability for users to set the relative priority of their own jobs and allow site administrators to control the priorities of different VOs.
• There is no effective control over bandwidth. The granularity of data handling (per job) does not provide full control over how many data transfers are currently running. Whenever any transfer is paused due to any reason (tape stage in, temporary unavailability), another transfer should use the available bandwidth. Dynamic re-distribution of bandwidth including pausing of low priority data transfers is needed to allow small data transfers to pass through while large data transfers are on-going.

• Lack of flexible assignment of credentials to data transfers. In the case of Grid users having multiple identities or belonging to multiple VOs it may become a challenge to combine data from different sources in one job. In the new architecture each transfer should be able to use its own credentials.

3. Related work
The importance of data movement in Grid and the need to treat data transfer activities as thoroughly as computational activities was already provisioned some time ago. I. Foster and K. Ranganathan proposed decoupling data transfer activities from computational activities, and for data transfers to be planned by a separate scheduler [7]. The computational scheduler, in turn, should use data location, availability, etc. while dispatching the jobs, i.e. be data-aware. The data problem is further explored and formalised by M. Allen and R. Wolski [8]. They present results of comprehensive research that undoubtedly shows an increased efficiency if data movement scheduling, even in its simplest form, is used. Other researchers have also proposed different algorithms to build data-aware schedulers [9].

Practical solutions to address data movement problems and perform data-aware scheduling have existed and been used in operational Grid systems for many years.

The gLite File Transfer Service (FTS) [10] reliably performs point-to-point transfers through the SRM protocol by creating a channel between endpoints. Channels are configurable and it is possible to configure for each channel a maximum number of transfers, allowed VOs, percentage share of simultaneous transfer for each VO and priorities of transfers within the VO, so the FTS combines reliable transfers with configuration flexibility.

Stork [11], [12] is a data transfer scheduler specially designed to implement reliable data transfer in a Grid independently from computational activities. Stork, a part of Condor [13], gives the ability to build a Grid infrastructure where data transfer activity is completely separated from computations, carried over by other modules of Condor.

Another possible way to address the data movement problem is to use replication and data driven scheduling, i.e. dispatching jobs to sites with data already available. The DIRAC [14] project uses this approach to evenly distribute data across the infrastructure and make it available instantly for the researchers. It must be admitted, however, that this approach is possible only in a fully centralised environment.

After detailed evaluation, it was found that none of the above solutions could be applied to solve transfer problems of ARC-based infrastructures. The FTS, though configurable and reliable, is highly centralised and does not fit into ARC’s design principles, and the support of only the SRM protocol also limits its usage. Stork performs reliable data transfer for many protocols, but requires a user or some high-level scheduler to plan the transfers. Moving jobs to data requires very sophisticated and effective replication and scheduling services, that are hard to implement in the decentralised ARC design.

4. Architecture
To satisfy the requirements outlined in Section 2, a three-layer architecture was conceived (see Figure 1). The unit of data transfer is a Data Transfer Request (DTR), which consists of a source endpoint, a destination endpoint and other transfer properties. The top layer (DTR Generator) is responsible for splitting job requests into file-level DTRs. It also takes care of
simplifying priorities to the level suitable for scheduling, assigns credentials and directs DTRs to the middle layer. It also accepts completed DTRs.

**Figure 1.** Three layers of data management architecture

The middle layer (DTR Scheduler) schedules individual transfers and suspends and resumes them, taking into account priorities, available bandwidth and current load. This layer is also responsible for processing pre- and post-transfer operations of data transfer protocols through DTR pre- and post-processor modules. For the physical data transfer the DTR is passed to the lower layer.

The lower layer (DTR Delivery) performs the actual delivery of data. It consists of multiple DTR delivery modules which implement reliable transfer and handle allocated bandwidth. In order to limit the resources consumed by the system the number of these modules is limited.

To illustrate the complexity of data processing involved, Figure 2 provides an example of processing a DTR for an input file for a particular job. Depending on the type of storage or catalog on which the data resides, various preparation and postprocessing steps may be needed.

**Figure 2.** States of data request processing

5. Implementation

It was decided to implement this new architecture inside the current ARC code base, rather than as a separate entity, in order to facilitate the
communication with the rest of ARC and to re-use as much as possible existing tried and tested code. ARC server-side code is written in C++ and C++ was chosen for the new code.

A DTR is the core data structure that includes all necessary information to perform a file transfer. Each DTR contains a source and destination endpoint (each having metadata where appropriate such as file size, checksum, creation date, etc.), transfer parameters and limits, credentials, local user information (UID/GID), Grid job ID, priority of transfer, and others.

The three layers corresponding to the Generator, Scheduler and Delivery are implemented as persistent threads within the main ARC server process. All DTR state is maintained in memory only, which avoids the need for an extra persistent layer such as a database to hold state information. In the event of a service crash or restart the DTRs can be recreated from information stored persistently such as files containing job descriptions. Pointers to the DTRs are passed around between components, which can modify them directly and push them between each other.

The task of the Scheduler is to send and receive DTRs to and from the other layers and run pre- and post-transfer operations, continuously prioritising the DTR queues. Therefore pre- and post-processor operations are performed asynchronously in separate threads spawned by the Scheduler. Physical data transfer in the Delivery layer is implemented in separate processes - the main reason for this is so that the local file system is accessed under the user account of the local user to which the grid user is mapped, rather than the user running the ARC service (usually root). Communication between the transfer processes and the main data staging process, for example to monitor transfer rates, is done through pipes on the local filesystem.

Priority handling is organised in terms of shares and priorities. DTRs belong to a transfer share, which can be defined as a share of available transfer slots per-user, per-VO, per-group inside a VO, or per-role inside a VO (these properties are defined by the Virtual Organisation Membership System (VOMS) [15]). The implementation of shares allows easy addition of different sharing criteria in the future.

Each specified share has a priority in the configuration. In addition, the user is able to specify their job’s priority in the job description. The Generator computes the final priority of the DTR from the priorities of the share and the job the DTR belongs to. At each iteration the Scheduler determines how many transfer slots every active share can grab according to its priority. Then the scheduler launches as many DTRs with highest priority in every share as there are free slots available for this share. This priority handling is illustrated by Figure 3, where shares are defined per-VO and VO2 has a higher share assigned. Queued jobs go to their share from the right of the queue.

| P=2 | P=5 | P=5 | P=8 |
|-----|-----|-----|-----|

| Queued slots | Running slots |
|--------------|---------------|
| VO1 share | VO2 share |

**Figure 3.** Illustration of priority handling. A higher value of P indicates a higher priority.

This algorithm allows dynamic adjustment of transfer slots per share, providing a single user with the whole transfer capability of the resource, but immediately restricting this user when other users start to use the resource. This algorithm also prevents the case where one user or VO could block another by submitting a large batch of jobs. At the same time the algorithm retains configuration flexibility, allowing resource administrators to tune the usage of the resource by different users/VOs and providing users with the ability to prioritise their jobs in the workflow.
6. Status, Future Work and Conclusion
At the time of writing a prototype implementation of the architecture described in this paper has been developed. It consists of a stand-alone data transfer scheduling system and has not yet been integrated into the ARC middleware. Only a basic priority system has been added, which has fixed sizes of transfer shares and simple highest priority goes first scheduling. However it is a fully functional system and contains all the features available in the current production version of ARC along with most features described in the requirements.

After rigorous testing to ensure the stability of this stand-alone system, it will be integrated with the rest of ARC, replacing the current data management system. As this is an internal component, there is no interface change and ARC users will not need to change any of their systems, but should notice an immediate benefit. Then there will follow a period of testing of the whole system with real workloads, during which the priority system will be tuned and various algorithms evaluated. It is expected that the completed data management system can be released into production in the second half of 2011.

This paper has outlined the problems experienced with ARC with the recent increased data management load and changes in access patterns. It has presented a new architecture designed to address these issues including advanced prioritisation algorithms. Future work will provide an evaluation of the benefits provided by the new system.

References
[1] Ellert M et al. 2007 Future Gener. Comput. Syst. 23 219–240 ISSN 0167-739X
[2] ATLAS Collaboration 2008 Journal of Instrumentation 3
[3] Jones R and Barberis D 2008 Journal of Physics 119
[4] Molfetas A et al. 2010 Proceedings of Computing in High Energy and Nuclear Physics (CHEP) 2010
[5] Sim A et al. 2008 The Storage Resource Manager Interface Specification Version 2.2 Tech. Rep. GFD.129
Open Grid Forum available at: http://www.ogf.org/documents/GFD.129.pdf
[6] I Foster C K and Tuecke S 2001 The Anatomy of the Grid. Enabling Scalable Virtual Organizations available at: http://www.globus.org/alliance/publications/papers/anatomy.pdf
[7] Ranganathan K and Foster I 2002 Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002, HPDC’02, IEEE Computer Society, Washington, DC, USA available at: http://www.globus.org/alliance/publications/papers/decouple.pdf
[8] Allen M and Wolski R 2003 Proceedings of the 2003 ACM/IEEE conference on Supercomputing, IEEE Computer Society, Washington, DC, USA available at: http://portal.acm.org/citation.cfm?id=1048935.1050193
[9] A Carpen-Amarie M A and Cristea V 2008 Proceedings of the International Workshop on High Performance Grid Middleware (HiPerGrid), Bucharest, Romania pp 33–40 available at: http://arxiv.org/abs/0901.0291v1
[10] Kunszt P et al. 2006 The gLite File Transfer Service available at: https://twiki.cscs.ch/twiki/pub/LCGTier2/FTSlinks/transfer.pdf
[11] Kosar T and Livny M 2004 Proceedings of the 24th International Conference on Distributed Computing Systems available at http://www.cs.wisc.edu/condor/stork/papers/stork-icdcs2004.pdf
[12] Kosar T and Balman M 2009 Future Gener. Comput. Syst. 25 406–413 ISSN 0167-739X available at: http://www.sciencedirect.com/
[13] Condor project Web site. URL http://www.cs.wisc.edu/condor/
[14] Tsaregorodtsev A et al. 2008 Journal of Physics: Conference Series 119, Part 6
[15] Alferi R et al. 2005 Future Gener. Comput. Syst. 21 549–558 ISSN 0167-739X