Evaluation of a new data staging framework for the ARC middleware

D Cameron¹, A Filipčič², D Karpenko¹ and A Konstantinov¹,³

¹ Department of Physics, University of Oslo, P.b. 1048 Blindern, N-0316 Oslo, Norway
² Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia
³ Vilnius University Institute of Applied Research, Sauliškio al. 9 III r., 10222 Vilnius, Lithuania

E-mail: d.g.cameron@fys.uio.no

Abstract. Staging data to and from remote storage services on the Grid for users’ jobs is a vital component of the ARC computing element. A new data staging framework for the computing element has recently been developed to address issues with the present framework, which has essentially remained unchanged since its original implementation 10 years ago. This new framework consists of an intelligent data transfer scheduler which handles priorities and fair-share, a rapid caching system, and the ability to delegate data transfer over multiple nodes to increase network throughput. This paper uses data from real user jobs running on production ARC sites to present an evaluation of the new framework. It is shown to make more efficient use of the available resources, reduce the overall time to run jobs, and avoid the problems seen with the previous simplistic scheduling system. In addition, its simple design coupled with intelligent logic provides greatly increased flexibility for site administrators, end users and future development.

1. Introduction

The Advanced Resource Connector (ARC) [1] is a Grid middleware which unifies a set of heterogeneous distributed computing resources into a single coherent resource. The ARC Computing Element is the component which connects each individual resource to the Grid. It acts as an interface between the resource and the rest of the Grid, allowing users to submit jobs which use the computing resource and publishing information on the resource and the jobs running on it back to the Grid. The most important part of the Computing Element is the ARC Resource-coupled EXecution service (A-REX). A-REX accepts requests containing a description of generic computational jobs and executes them in the underlying local batch system. Typically, these jobs require input data to process and produce output data with results. A-REX takes care of staging in files containing input data or program modules from a wide range of potential sources before submitting the job to the local batch system and storing or staging out the output results after the job has completed.

The data files specified by the user may be “physical” URLs which give a direct link to the data, “meta” URLs which involve some preparation of or redirection to a physical URL, or “index” URLs which refer to files in a catalog. The catalog may map files to several meta and/or physical URLs representing distinct replicas of each file. In addition, A-REX has the ability to cache input data on a local file system so that repeated requests for the same files can...
use a local copy instead of repeatedly downloading from remote storage. There can therefore be several operations to perform before and after the physical transfer takes place.

With highly data-intensive jobs, A-REX data staging becomes a critical factor in the performance of a Grid site. Because all data for all jobs on the site flows through a single point, the data staging system must be efficient at processing the incoming flow of jobs, as well as having high performance physical capabilities such as network bandwidth and disk storage. Over recent years the data demands of jobs have increased, and limitations became apparent in A-REX’s original data staging system. This led to the creation of a new data staging system [2]. The rest of this paper describes the implementation of this system and presents an evaluation of it compared to the old system. Section 2 gives a detailed description of the design of the new system and lists the ways in which it performs better than the old system. The implementation details are explained in Section 3 and results are presented in Section 4.

2. Design of the New System

Although A-REX’s old data staging system had proved reliable for several years, it had a rather rigid framework which could not easily be adapted to an increasing demand from data-intensive jobs [2]. A fixed number of data staging slots were assigned to jobs and each file within a job was processed sequentially. This created the problem that a file blocked in a preparation stage would waste network resources, and a job with one file blocked could stop a new job from starting. Additionally, jobs where all files were cached had to wait behind other jobs which had to do data transfer. This led to the creation of an entirely new design for data staging, shown in Figure 1.

![Figure 1. New data staging framework, based on file-level granularity and three separate layers of processing.](image)

There are three layers in the new system. The Generator (upper layer) uses user input of jobs to construct a Data Transfer Request (DTR) per file that needs to be transferred. These DTRs are sent to the Scheduler (middle layer) for processing. The Scheduler sends DTRs to the Pre-processor for anything that needs to be done up until the physical transfer takes place. This can involve checking the cache for the presence of the file, resolving replicas from index URLs, checking the source file exists, deleting the destination in the case of overwriting...
and obtaining physical URLs if a meta URL used. Then the DTR is sent to Delivery (lower layer) for the transfer itself. Once the transfer has finished the Post-processor handles any post-transfer operations such as releasing requests, registering new replicas or processing cache files. The number of slots available for each component is limited, so the Scheduler controls queues and decides when to allocate slots to specific DTRs, based on the prioritisation algorithm implemented.

This design has several benefits:

- Separation of operations into different components means that these operations are independent rather than serialised, so a pre- or post-processor operation cannot block a DTR in another operation.
- Cached files can go straight from pre-processor to post-processor, skipping the delivery queue.
- Files stored on low latency media such as tapes are staged to disk asynchronously using protocols such as SRM [3]. The staging request is made and then polled at intervals, allowing other DTRs to be processed in the meantime.
- A site-wide queue allows multiple DTRs from different jobs (sharing the same credentials) to be combined in bulk operations for certain pre- and post-processor steps such as resolving replicas.
- On cancellation of one or more DTRs or jobs (or all DTRs in the case of A-REX shutdown), each DTR can be safely stopped, any unfinished transfers can be cleaned up and resources such as cache locks can be properly released.

The new system also introduces the idea of priorities for data transfer. The old system used a fair-sharing mechanism to assign a transfer share to each job, based on certain attributes of the job owner, such as Distinguished Name (DN) or Virtual Organisation (VO) they belong to. The available transfer slots were then split among the active transfer shares, which meant that one user or VO could not block other users in the queue by submitting a large number of jobs which occupied all the transfer slots. Although this mechanism solved some issues, it did not allow users to specify inside their share which jobs were more important than others and hence should be processed faster.

The new system maintains the transfer shares concept, but also solves the latter problem through the use of two configurable properties: user-defined job priority and server-defined share priority. Priority is defined as an integer between 1 and 100 inclusive where a higher number is a higher priority. Users may define a priority for their jobs in the job description and this is a measure of the priority given to the DTRs for this job within the transfer share it gets assigned to when submitted. On the server-side, it is possible to define a share type, and the base priorities of certain shares. The base share priority is used to determine the number of transfer slots to assign to each share when transfer slots are divided among the active shares. Thus service administrators can set maximum priority limits for certain shares, but users or VOAs have full control of their jobs’ priorities within the share.

While revising the Delivery and Processor queues, the Scheduler separates DTRs according to the shares they belong to. Inside every share DTRs are sorted according to their priorities. Then the Scheduler determines the number of slots that every active share can use, depending on the priorities of active shares and launches this number of DTRs from the end of the queue.

3. Implementation
The lower two layers of the new system are implemented as a separate library, independent of A-REX or other ARC services, meaning it could be used by any generic data transfer system. An API allows any Generator to submit to and receive DTRs from the Scheduler.
The Generator implemented in A-REX performs the task of taking submitted job descriptions and sending the input and output file requirements to the Scheduler before and after batch job submission. The Scheduler, Delivery and Processor components are implemented as threads and they communicate by passing DTR objects held in memory. The physical transfer is done in a separate process for each DTR, mainly because the data transfer may need to run under a separate user from the user running A-REX. The processes communicate transfer progress information back to the system by sending serialised objects through stdout.

Holding the state of the system in memory allows extremely fast processing without the need for an external component to keep persistent state. It also makes communication trivial as objects can simply be passed from one thread to another. However, in the event of an unexpected crash or power cut, all in-memory state will be lost and transfers could be left half-finished. In order to reduce this problem, the state of all DTRs is periodically dumped to a file. On restart, the contents of this file are read and any half-finished transfers are cleaned up before being retried. The file also acts as a source of information for monitoring of the system by GangliARC [4].

Since all data staging in ARC is performed by a single host, that host can become the bottleneck for the whole site. One way to ease the bottleneck is to allow other hosts to share the load of data transfer, specifically the network load, in order to give an increased effective network capacity. The new system achieves this by allowing the Delivery layer to be spread across multiple hosts, while still keeping the Scheduler logic on the main A-REX host, shown in Figure 2.

**Figure 2.** Data staging over multiple hosts.

A DataDelivery Service is set up on each host which should perform data staging, and this service is registered with the Scheduler. When a DTR is ready for Delivery, the Scheduler can choose the host on which to run the transfer. If a remote host is chosen then the Scheduler calls the DataDelivery service on that host to request it to run the transfer. The service acts like a simple Scheduler which immediately starts Delivery processes for any requests it receives. The Scheduler regularly polls the service for the status of each DTR. The advantage of this system
is that if a service on any remote host stops or the host goes down, the Scheduler can simply retry on another host. Setting up and configuring the remote hosts is also extremely easy, since all the intelligence and logic is in the Scheduler on the main A-REX host. This architecture in addition allows special situations such as remote hosts each having a cache on local disk — in this case the Scheduler can direct a DTR which is writing to a cache on the corresponding remote host.

4. Results
To test the effectiveness of the new framework, several performance tests were carried out comparing it against the previous framework. The tests consisted of submitting a job to an ARC CE with various data requirements and measuring the time for the job to complete. The jobs were done under controlled conditions to minimise any external effects on the results, but realistic workloads were used. The job itself did not perform any processing, and so the measured job completion times depend only on the time to perform data staging before and after the job. All tests started with an empty cache.

The test environment consisted of a dual-core desktop PC with ARC clients and an ARC Computing Element from ARC version 2.0.0 and cache and session (jobs’ working) directories on a local disk. The PC was dedicated to running these tests and did not run other jobs. It was configured to have the same parameters for old and new frameworks, with respectively 8 threads per loader process and 8 delivery slots.

The data requirement for each job was one of four datasets from the ATLAS high-energy physics experiment [5] stored in the Nordic ATLAS cloud and each file in the dataset was specified in the job description as a LFC URL. A variety of different dataset and file sizes were used, shown in Table 1.

| Dataset Name | Number of files | Average file size | Total size |
|--------------|----------------|------------------|------------|
| Dataset A    | 600            | 500KB            | 320MB      |
| Dataset B    | 39             | 15MB             | 622MB      |
| Dataset C    | 200            | 50MB             | 10GB       |
| Dataset D    | 20             | 1.5GB            | 29GB       |

Table 1. Content of sample datasets

Within each dataset the file sizes were roughly the same, except Dataset C where file size varied between 13KB and 100MB. The average size of all ATLAS data files is around 320MB but with a very wide distribution [6]. Whilst the datasets with large numbers of files are not typical of real ATLAS jobs, they provide a convenient way to compare the effects of the two data staging frameworks under high load. It can happen in real cases that when a large number of jobs are submitted in a short period of time there are several thousand files in the queue. Each test was repeated 5 times with new and old data staging and the average job times along with their standard deviation are shown in Figure 3.

The first thing to notice is that for small files, the number of files to transfer has much more of an influence on the job time than the total amount of data. For both old and new data staging, it is much faster to transfer Dataset B than Dataset A, even though there is twice as much data to copy. This is because the overhead of operations before and after the transfer is much larger than the time to do the physical transfer. These operations include checking and locking the cache, looking up replicas in LFC, and preparing and releasing TURLs from SRM. The huge difference between old and new data staging for Dataset A can be explained by this overhead. With old staging the 8 available slots spend most of the time doing the pre- and post-processing and very little time doing the physical transfer. In contrast the new staging separates the pre-
and post-processing from the transfer, meaning that the physical transfer is done in parallel and so all 8 slots are only doing transfers. In addition the bulk operations used by the new staging mean that pre-processing operations are done orders of magnitude faster and so there is always a full queue of DTRs waiting for physical transfer.

As the file sizes get larger, the ratio of time spent in physical transfer to other operations increases, and the advantage of the new staging is less. With the largest files, in Dataset D, the job times are the same (within error) for old and new staging. It can also be seen that the standard deviation increases with file size, since fluctuations in network traffic have more of an influence (the time for pre- and post-processor operations is relatively constant). An obvious point to make is that with either system, having larger file sizes leads to a massive improvement in processing time.

The difference between the old and new systems is clearly illustrated in Figure 4, which shows the incoming network traffic during the transfer of Dataset C with new and old staging. The rate is around 50% higher with new staging, because there are continuously 8 active physical transfers ongoing, whereas with old staging some of the 8 available slots are taken up with other operations.

Figure 5 shows the effect of the priority system in the new staging. 100 jobs were submitted, which required a random set of between 1 and 20 input files taken from a dataset of 1200 files,
and uploaded 1 output file. This meant that the same files could be required by several jobs. A priority of either 50 or 100 (selected randomly) was given to the job. It can be clearly seen that the high priority jobs were processed much faster than the low priority jobs. The average time for a high priority job was 1696 seconds and for a low priority job 2661 seconds. The tail of long high priority job times was mainly caused by low priority jobs requiring the same files as those jobs and holding cache locks for them, blocking the high priority jobs. This issue will be addressed in future work to optimise the new system.

5. Conclusion
In this paper a new data staging system for the ARC Computing Element has been demonstrated to give a large improvement in job throughput, due to more efficient use of network resources and intelligent transfer scheduling including the separation of physical transfer from other operations. The system also provides an easy method for scaling out data transfer to multiple nodes which further increases the amount of data that can be transferred. In addition, the introduction of priorities and shares brings flexibility for both users and administrators. Future work will consist of optimisation of the priority system and further development based on feedback from users.

References
[1] Ellert M et al. 2007 Future Gener. Comput. Syst. 23 219–240 ISSN 0167-739X
[2] Cameron D, Gholami A, Karpenko D and Konstantinov A 2011 J. Phys.: Conf. Ser. 331 062006
[3] Sim A et al. 2008 Tech. Rep. GFD.129 Open Grid Forum available at: http://www.ogf.org/documents/GFD.129.pdf
[4] Cameron D and Karpenko D 2012 Proc. Computing in High Energy Physics 2012 (CHEP 2012)
[5] ATLAS Collaboration 2008 Journal of Instrumentation 3
[6] Zang D et al. 2012 Proc. Computing in High Energy Physics 2012 (CHEP 2012)