Data Handling Requirements for Intensity Frontier Experiments

CD/REX department

v1.0

19 October 2010

Introduction

Data handling means the problem of uniting the appropriate input data files with running batch jobs, and returning the output files to the correct storage area. The complexity of this task depends both on the amount of data produced by an experiment, and the geographic diversity of its operations; efficiently running high throughput jobs on widely dispersed grid computing elements requires a significantly more advanced infrastructure than running at a single site that is co-located with the entire dataset.

For the purposes of this document “data file” refers to the physics data produced by the detectors and any reconstruction and analysis applications. It does not include such things as executable programs, calibrations and other run dependent data, and the batch job log files, all of which are covered by other requirements documents. However, in some cases the methods used to transfer data files can also be used for these types of file

Basic requirements

The requirements for jobs which process data files can be split up into several basic stages:

1. Identify the input dataset. Generally the files which need processing share common characteristics which identify them. For example, for raw data this could be the run number or time it was taken.
2. Locate the files within the dataset. Each file will be available from one or more storage systems.
3. Bring the data to the job (or vice versa). The job and the data files it needs are brought together to perform the processing.
4. Return the output file to the appropriate place. For reconstruction jobs this may mean a permanent storage location, while for user analysis this may be a transient file that only needs to be retained until the user collects it.

Storage interfaces and transfer protocols

The jobs and data handling system will have to handle a variety of different storage systems. This is not intended as an overview of the different technologies available, but rather of the standard interfaces to them.

Worker node file access

Normally, individual jobs will be required to copy the input file from the storage element to scratch space on the worker node. Copying the data reads large consecutive blocks from the storage system, while reconstruction and analysis processes tend to read small chunks of the file, which puts more load on the storage. Interactive users and small numbers of batch jobs can access files directly, but large numbers of jobs slowly accessing the shared storage can reduce performance. Additionally, it is usually necessary to limit the number of simultaneous transfers. Some storage systems can do this themselves; those that don't may need an external solution.

Once the job has finished with the file, it should delete the local copy.

SRM (Storage Resource Management)

SRM is not a storage type itself, but a standardized grid storage access interface, which provides a common interface for multiple storage types, and largely allows clients to treat grid storage elements uniformly, regardless of the actual technology used to implement them. SRM allows clients to request file transfers in and out of the storage element, plus provides management commands to do things like creating directories and listing their contents. The SRM server does not perform the actual data transfer itself; instead it delegates this to a protocol such as GridFTP. As the standard grid interface, SRM is something jobs running on grid sites will have to use to perform file transfers.

GridFTP

GridFTP is the standard grid file transfer protocol, designed to improve the performance of transfers over high latency network connections. GridFTP does not provide any load balancing or throttling controls: these are more properly handled by the SRM layer.

POSIX filesystem interface

Some storage systems (e.g. NAS systems like BlueArc) can be mounted directly on worker nodes. Where such systems are known to be available it is possible to avoid some of the overheads of using the SRM interface, instead accessing the data directly through the file system. However, there is usually no load control available when doing this, so some extra method of limiting the number of simultaneous users may be required.

Multi-tier storage

Multi-tier storage means a storage system with multiple tiers which have different performance characteristics. Examples are things like tape backed disk caches or NAS systems with fast and slow disks (or in the long term future disks and flash storage) where the files can be migrated between the two types. Depending on the technology used, muti-tier storage may need special handling to make the most efficient use of it. Even if the storage system provides a uniform namespace for its contents, the performance of accessing different files within it may vary wildly. Especially where the latency of the slow storage is poor, prestaging the desired files into fast storage may be necessary to reduce idle jobs.

More detailed requirements

Identifying the input dataset

This stage normally requires some sort of *metadata catalogue*, which stores information about the

contents of the data files, and a method of querying the catalogue to obtain a list of files matching whatever criteria the job needs. The metadata catalogue can also be used to store information on the lineage and processing history of the files.

Locating the data files

This requires a *location catalogue*, which stores the locations of all data files tracked by the experiment. Using the catalogue the physical location of any requested file can be obtained. To be useful, the catalogue needs to be kept reasonably up to date, so it must be integrated with any file transfer service.

Requesting files by name or by dataset

A simple model is for the jobs to request files by their name – the job obtains (either as part of the submission stage, or when it runs) the list of wanted files and locations, and then ask for those files in order by the provided location. This gives predictable and relatively simple behavior, but is not necessarily the most efficient way of doing things.

An alternative is for the job to request the entire dataset from the data handling system, and let that return the file locations in the order it chooses. This brings benefits if the system has extra knowledge about the which files are not immediately available. Then it can notify the job with the files that are immediately available, while requesting the unavailable ones.

A further benefit is the case where a dataset is too large to be processed by a single job. In the simple model it is the responsibility of the submitter to break the large dataset into smaller pieces. With the alternative, a set of multiple jobs can request files from the same dataset, and the service is responsible for assigning an unprocessed file from the dataset to a waiting job as needed. Especially if there is significant variation between processing times of different files then this can provide an efficient assignment of work to the individual jobs.

For some experiments it may be necessary to process linked files at the same time (for example, near and far detector). This can be easily handled by the simple approach, but is more complicated where the data handling system determines the processing order.

Bringing the data file and job together

Where jobs are being run on dispersed grid execution sites there are two main approaches to the problem of getting the job and data together (alternatively, the jobs can fetch the files to the worker nodes direct from a central location, skipping any local site storage, but this is likely to prove inefficient for more than minimal data volumes). One is to bring the job to the data, the other is to bring the data to the job. The former requires more administrative control and requires the experiment to create carefully defined datasets, the latter provides more flexibility, but may be less efficient.

With the job to data approach, a limited number of people are responsible for determining which dataset should be located on which storage element, and on how to manage the space assigned to their VO. For more than the smallest volumes of data, an automated file transfer service is required to transfer the files around, and delete those that are no longer needed, but this is something that is only required intermittently. The job submission process needs to be integrated with the data handling system in order to direct the jobs to execution sites which contain the required dataset for the job.

For the data to job method, a file transfer service copies the files to the local storage element on demand, and notifies the job when the file is available. If old files are removed on a least recently used

basis, this will keep popular files available by erasing unused ones to free up space. No administrative control is required, but if the available space is less than needed by the datasets than are commonly used, it can lead to an inefficient cycling of files in and out of the storage element. This method does not require integration with the job submission process, but this may still be beneficial as it is preferable to send jobs to locations where much or all of their required datasets are already available.

Pre-staging files

Where files are being copied to a storage element, or where they are located on a multi-tier storage system, the efficiency of the jobs can be improved by pre-staging the files before running the jobs. This can be done manually by the job submitter as a separate step, or automatically by delaying the submission or execution of the jobs until some or all of the files are available in immediately available storage. The latter requires intergration with the job submission system.

Ad-hoc file transfers

It may be necessary to transfer files that may not included in the main metadata and file location catalogues. Such files can include things like the intermediate stages of the reconstruction program, and job output files. The file transfer mechanism should be able to move such files around.

Returning job output

Grid jobs run using shared accounts. When job output files are returned to the submitter they may be in a shared temporary area owned by a group account. There need to be tools to move the files to their final location and ensure they have the correct ownership.

Client interfaces

To enable maximum integration with job submissions, batch scripts, and experiment frameworks, the client side interface to the data handling system should be callable from, at least, shell scripts, Python, and C++.

Monitoring

Each job should provide notifications of its data handling activities to the monitoring system – requesting next file, starting to transfer file to worker node, finished transferring file, etc. This helps with pinpointing problems where jobs get stuck waiting for files.