Fermilab FERMILAB-TM-2192 December 2002

November 7, 2002

I. Mandrichenko, I. Terekhov, F. Würthwein

Run II Data Analysis on the Grid

Components and Interfaces

Abstract

In this document, we begin the technical design for the distributed RunII computing for CDF and D0. The present paper defines the three components of the data handling area of Run II computing, namely the Data Handling System, the Storage System and the Application. We outline their functionality and interaction between them. We identify necessary and desirable elements of the interfaces.

1 Introduction and Motivation

Computing at D0 and CDF (as is the case with most HEP experiments) is centered around data processing. Consequently, the meta-computing systems of the two experiments are expected to be built around data access and data handling. Since the experiments' data processing applications are fundamentally similar, (even though their frameworks and data access infra-structures differ), the experiments gravitate towards similar systems such as SAM and Enstore-dCache.

The two experiments early realized the importance of data handling and have been devoting considerable effort to building and adopting data handling and storage systems. In the Grid era, however, both applications and storage systems are evolving towards network-capable data access. In addition, object-level (as opposed to file-level) data access becomes popular in the Grid world. Relative to the initial requirements and existing design, the principal roles of the data handling components are changing as well. Thus, in order to architect a true Grid computing system for the experiments, it is imperative to develop a modern, Grid perspective onto the Data Handling for the experiments. We need to generalize the known concepts so as to include the new, emerging technologies, and make a clear distinction of the principal responsibilities from desired features.

2 Components

In the present paper, we define the three high-level components of the Grid architecture involved in data processing, see Figure 1.

- Application - user program designed to access (consume and/or produce or simply move) data somewhere on the Grid. As a result of data processing, Application may produce some new data and store it so that it will be accessible later by some other application, and/or be "extracted" out of the grid environment upon completion to a user specified location.
- Storage System - the entity which is responsible for storing data and providing access to it to the Grid. Storage System is assumed to be associated with one of Grid clusters. Storage systems span a wide range from "heavyweight" robotic storage to "lightweight" local disk. It is essential for the Grid that these systems share a well defined minimal set of common interfaces that are based on general protocols.
- Data Handling System - component of the Grid software responsible for maintaining data on the Grid and delivering the data to the Application. It is assumed that one of important functions of Data Handling system is to optimize access to the data replicating data and moving data item replicas from one Storage System to another as necessary.

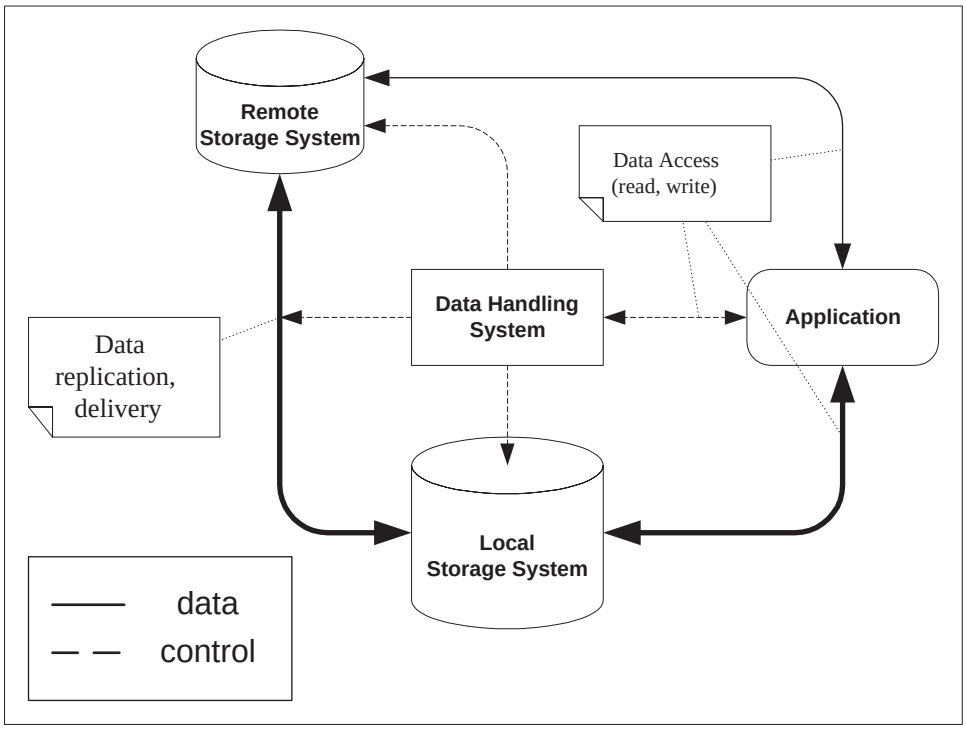

Figure 1: The client-server relations among the components and data movement paths.

We define the three components in some detail, based on their properties and roles. Furthermore, we define the high-level interfaces between them.

2.1 The Application

Before the application can run, it has to declare that it needs to process a set of data items identified in terms of the experiment's *Metadata Name Space* (MDNS). In addition, it announces its capabilities to access data in terms of protocols that its I/O module supports. For example, an application that is only capable to read/write files from/to a locally mounted disk will be distinguished from an application with RFIO support compiled into it. To be precise, these announcements may actually be made in the job description, but this is outside the scope of this document.

For input data, the URL's expected by the applications point to a readable area in a "local" storage system. Ideally, output data is handled symmetrically whereby the application first receives a URL pointing to a writable area in storage system before sending there the produced data. In practice, however, it is necessary to support the mode where the application first writes data to a "scratch area", i.e., to a volatile area not managed by the DHS, and upon completion the DHS is requested to import the data file. In either case, the application is responsible to generate meta-data describing the produced data, details

and extent depending on the experiments.

2.2 The Storage System

As the name implies, Storage System is responsible for storing data. Some Storage Systems will be located, networking sense, *near* one of the data processing Clusters and therefore data stored there will be *locally* accessible by the Application running on that Cluster. Other Storage Systems will provide remote access to a variety of Clusters. The third class of Storage Systems will only be available to the Data Handling System to retrieve files to a remote "cache" (another Storage System). In any case, the access is done by a logical URL such as the "/pnfs" path in Enstore.

The lifetime of a Storage System, and therefore its contents, is not constrained by that of a job or affected by the job scheduler's decisions. Thus, a *sandbox* created for a Grid job for the duration of its execution is *not* a Storage System.

We assume that the contents of the Storage System, as well as information about access rights etc., are publishable via the *meta-data catalogue* (Section 2.3) of the associated experiment, i.e., the Storage Systems are *public*. The project disks at D0 may be semi-permanent but are private and therefore are not considered Storage System.

We recognize two major types of Storage Systems:

- **Permanent Storage.** Integrity of data stored in a permanent storage is assumed to be "permanently" maintained by such storage system. Usually, tape-based systems such as HPSS or Enstore (with dCache).
- **Temporary Storage** is used to store data for a limited time. Usually this is the time required for one or more batch jobs to process the data. Also, temporary storage can be used by the application to store produced output before the data is moved to more permanent storage by the DHS. Usually, disk-based systems and systems with limited capacity such as Disk Farm or SAM cache on a file server node are considered as temporary storage.

It is not always possible to clearly distinguish between these two types of storage systems, and phylosophically, there is no such thing as "permanent" storage, and therefore there is no clear definition of terms "permanent" and "temporary". Also, *permanency* of particular storage system instance is largely determined by local support policies and the role the instance plays in the overall system architecture rather than by the nature of the storage system. However, for practical purposes, it is convenient to draw the distinction between these two types and expect them to have slightly different functionality as described in 3.1.

Storage Systems will vary in complexity from e.g., a single UNIX disk to e.g., an Enstore system with dCache. A Storage System may "have a life in its own" such that it may well decide to rearrange internal placement of the data (e.g., move data from its disk cache to tape). In doing so, however, the Storage System is expected *not to compromise* external access, i.e., it must maintain the validity of the replica catalogue maintained by the Data Handling System. In particular, a Storage System will not permanently erase a data file unless instructed to do so by the Data Handling.

A Storage System is typically protected by security, including Authentication, Authorization and Accounting policies. Security is a property of the Storage System that exists in addition to any security properties of the Data Handling. For example, a Storage System may require a valid AFS token whereas the Data Handling system requires a valid GSI proxy certificate. The credential used for the actual storage access may be that of the initiating user or of the Data Handling System, in which case the Storage System must be able to recognize the Data Handling as an entity (e.g. by a generic UNIX account such as "sam" or by a *service* Kerberos principal).

Naturally, we desire Storage Systems to perform internal optimizations of the data requests *known to it*. The Storage System will *not* attempt to optimize *planned* requests that are known to the Job Scheduler and/or the Data Handling but not to the Storage System. It is generally desirable for the Storage System to cooperate with the Data Handling to optimize data access beyond currently active requests (i.e., with application data access in progress). To achieve this goal, the Storage System may either provide reservation services (or otherwise be able to receive notification of future data requests), or expose to the Data Handling the *economics* of its internal data placement. The latter is exemplified by Enstore which uses tapes for ultimate storage; the costs of reading from a mounted and unmounted tape volumes differ dramatically and therefore Enstore exports file placement by tape to the SAM system which can order accesses to Enstore in "space" (e.g by volume) and/or in "time" (e.g. the whole tape at a time).

Thus, Storage System, if applicable and possible, may provide *resource management* services. These are not for mere optimization; in the case of Enstore for D0 it was in fact *necessary* to design resource optimization (including volume assignment exposure) into the overall D0 data handling system, based on the calculations that showed that unordered tape access by the experiment would exceed the capacity of the tape-mounting robot.

2.3 The Data Handling Per Se

The Data Handling System (DHS) primarily maintains the meta-data name space (MDNS) by allowing the Applications to create (and possibly remove) entries therein. The DHS further provides the service of management of logical groupings of such entgries, by means of *datasets*. The MDNS entries are data files or objects that are associated with the description of the contained data (this is highly experiment-specific) as well as with physical locations

in one or more storage systems (experiment-independent). The physical locations as viewed by the Data Handling System are in turn logical URL's when seen by the Storage Systems, as we explain below.

To retrieve input data, Application issues request to translate a set of entries in the MDNS into a set of URL's. Along with the identification of the required data items, Application provides list of protocols it is capable of using to retrieve the data. In a pure sequential model, these translations are done one at a time in the order decided by the DHS. This is the first and foremost service provided by the DHS to the Application. One or more URLs returned to the Application consist of:

- data transfer protocol (to date, perhaps the most common has been "local")
- Storage System identification (perhaps in the form of "head" host address)
- Logical path to the data inside the Storage System

It is responsibility of the DHS to deliver data to one of Storage Systems accessible by the Application (as described by the list of acceptable protocols) and, if possible, to optimize data access by delivering the data *as close as possible* to the Application. Application's request for data translation blocks until the requested data item is delivered. Typically, the Storage Systems accessible to the Applications running at a Cluster are different from those that actually contain the data. In these cases, the service provided by the DHS is in moving the data "closer" to the application, i.e. from "remote" Storage System to a "local" one. Such a movement is referred to as *caching*. Please note that the Storage System acting as a local cache *is not* a part of the DHS (see 1).

Implicit in the above described service of the DHS is the ability to not only recognize various protocols supported by the application, but also relative cost for using them (see the resource management discussion later).

When the Application stores data, the DHS service is largely symmetrical. Generally Grid Application is not expected to know what cluster it is going to start at, and therefore, what Storage Systems are accessible from there. That is why before the Application can store the data, it requests the list of Storage System identifications available based on the list of acceptable protocols. DHS returns such a list of Storage Systems with relative preferences of using each of them. Among other factors, preferences will be based on proximity of the Storage Systems and perhaps on their availability.

Application will choose one of Storage Systems returned by the DHS and deposit the data item into the Storage System. Then it will notify DHS that new data item identified by its MDNS ID is created and provide its URL in the Storage System.

Upon notification, the DHS will verify the meta-data (especially when the produced data is intended to be shared by other collaborators) and if necessary initiate moving of the

data further to a final ("permanent") Storage System, based on the Application's preferences and policies of the experiment.

An alternative way of storing the data is as follows. The Application may first create an output file in a private area such as its sandbox, and after that issue a request to store the file with a Storage System. The Application must wait until the DHS has reliably stored the data into a Storage System (which need not be the final destination) and may want to be notified when the data has reached the final destination Storage System.

Additional services are highly desirable although not critically necessary and are irrespective of read vs. write; these are reliability, security and resource management. Reliability features can be divided into those concerning a single Storage System and those of overall nature. For a specific Storage System, the DHS will in general have another level of retrials and timeouts when accessing the Storage System (on top of what it may already provide). For the overall robustness, an intelligent DHS will automatically resort to alternative replicas when the first replica attempted has failed.

For the resource management, the DHS will provide:

- Optimization of access to an individual Storage System by planning the requests for it;
- Replica Selection in order to choose an optimal source or destination of the data, based on the configured cost metric;
- Coordination of read access to a "local" Storage System by consumers, so as to minimize cache turnover rate.

For security, the DHS will provide translation between the AAA characteristics of individual Storage Systems and the experiment's *Virtual Organization* as a whole, i.e., it will either use delegated user's credentials, or present its own, when accessing a Storage System. Thus, a Data Handling System may be a secured resource in itself, just like Storage Systems always are.

We note in passing that security and resource management are often put into the broader category of *policies*. We separate the two classes of services, however, because resource management may become a critical, rather than a desirable, service (Section 2.2) of the DHS that no other entity can provide.

3 Interfaces

Please refer to Figure 1 for our client-server model. In this Section, we describe the interfaces of the two server components, the Storage System and the Data Handling System.

3.1 Storage System Interface

Storage systems are always servers, never clients in the context of data movement. They neither initiate contact with data handling nor with the user application.

Storage System interface is expected to provide at least the following functions:

- File-level access to the stored data with semantics similar to one of UNIX File System: cp (get/put), ls. This functionality is required of any Storage System. Also, *temporary* storage systems, and systems with limited capacity are expected to provide functionality to remove data and release space occupied by the data and to query space availability.
- Record-level access as a subset of POSIX API: open(), read(), write(), seek(), close() functions. This suit of functions seems to be highly desirable, but not always necessary for the Storage System to provide.
- A storage system may provide multiple types of interfaces that implement the above, and distinguish themselves in the level of "persistency" or "permanence" of the stored data. The level of permanence of each of them needs to be exposed via the interface.
- Depending on the nature of the Storage System, when applicable, it may expose some resource management functionality via its interface. Example functionalities are advance space reservation and file locking/unlocking. However, in the Stage 1 of the project it seems to be useful and reasonable to assume that Storage Systems provide abundant resources so that such functionality is not required.

We explicitly note that a complex storage system may internally use DH functionality like MDNS as well as replication in order to fully integrate static as well as dynamic caches with robotic storage, for example. It is up to the local owner/operator of a given storage facility to decide to what extent they are interested in exposing modular components of their complex storage system to the global data handling, or expose it only as an integrated system.

3.2 Data Handling System Interface

Data Handling System (DH) is supposed to be a client of Storage Systems and act as a server for the Application. Four major functions of DH exposed through its interface are:

- DH maintains meta-data name space (MDNS) by allowing the Application to create new data items and data sets and enter them along with their physical location into MDNS by notifying DHS. It is assumed that before making new entry, the Application delivers the data item into one of Storage Systems, known to DHS and then gives DHS URL pointing to the location of the data item.
- DH serves Application request to deliver input data by translating data item identification expressed in terms of MDNS into a URL referencing actual location of the data file in one or more Storage Systems.
- Application is supposed to notify DHS when requested data item, previously delivered by DHS to the application is not needed any more. Until then, the data item will be considered by DHS as "locked" and will not be removed by DHS. However, if the application fails to notify DHS the data will be considered "released" by DHS after some reasonably long time-out interval.
- DHS accepts Application requests for data movement between Storage Systems.

4 Summary

Based on our experience and understanding of current D0 and CDF computing as well as their forseeable development into Grid era, we have analysed data handling process. We have identified three major components of Run II grid computing involved in data handling: Data Handling System, Storage System and Application. We have presenetd high level overview of their functionality, interaction between these components and outlined major elements of their interfaces. We identified necessary and desirable elements of components interfaces. Conceptually, most important difference between the earlier views of Run II computing and what is presented here is that the Data Handling System is now decoupled from what was previously considered to be its internal cache. Such cache is now viewed as a specific case of a Storage System. We believe this makes the functionality of DHS more clearly defined and therefore allows for simpler design of DHS.