# Managing Small Files in Mass Storage Systems using Virtual Volumes

Esther Acción, Manuel Delfino*
Port d'Informació Científica, Spain
Emilio Hernández†
Universidad Simón Bolívar, Venezuela
Andreu Pacheco‡
Institut de Física d'Altes Energies, Spain

*Abstract*

We present a storage middleware, called ViVo (Virtual Volumes), for grouping files into volumes transparently in a Mass Storage System (MSS). When a user requests a file through any file transfer service, such as ftp or http, the ViVo middleware is triggered. ViVo reads the volume from the MSS and mounts it in such a way that the file service daemon can read the file and serve the user request. Storing files in volumes, rather than storing them directly, is important for the efficient use of the MSS, especially if it has a long access latency. This is the case of most MSS systems, such as tape robots, in which the requests are queued until a tape drive is available. Data access efficiency can be increased by reclustering files into different volumes for improving locality of reference.

## INTRODUCTION

Access to large data repositories in distributed environments such as a Grid, poses problems related to efficient data storage and transmission. In particular, efficient hierarchical storage management of a large amount of small size files continues to be a challenge. Storing such files directly on tape-based tertiary storage leads to extremely low operational efficiencies. Commercial tape virtualization products are few, expensive and only proven in mainframe environments. Asking the users to deal with the problem by bundling their files leads to a plethora of solutions with high maintenance costs. Part of the problem is that data processing environments have evolved towards the illusion of an infinite file store with a subdirectory structure, eliminating the concept of a volume, be it physical or logical.

The file size is an important factor for achieving a good throughput from "nearline" platforms such as tape based MSS. The reason is that there is normally a long delay to start transmission, due to queueing and mechanical tape access and placement in drives. There have been studies that relate the file size to tape drive efficiency, which conclude that small files dramatically downgrade the efficiency of tape usage [1]. The main problem is the initial delay for the actual file read and for this reason we may expect a similar behaviour on any system with similar characteristics, that is a system in which queueing time dominates the data retrieval.

There are file migration systems oriented to MSS systems, such as Castor [2], Enstore [3] and HPSS [4], among others. These systems are typically designed for moving files across a storage hierarchy on request. When used with large collections of small files the access to the drives of the back-end becomes congested, generating long query queues and affecting other MSS users. The user manuals of these systems invariably suggest the use of tar archives when many small files are used. There are also client-server systems for accessing distributed file systems, such as SRB [5] and GFAL [6] that need the applications to be linked with special libraries for remote file access.

We intend to develop mechanisms for transparent volume creation and access that do not need to modify the applications at all, not even recompile them or relink them with a special wrapper library. We propose a two-level storage and transfer mechanism, useful for MSS as well as distributed systems such as Data Grids. Data movement at the first level is made on a file basis, while data movement at the second level is made on a volume basis, where a volume is a collection of files. In this paper we present a mechanism that follows this scheme under a specific but commonly used producer-consumer scheme.

The rest of this article is divided as follows. First, the data flow model is described, as well as the mechanism for volume creation and manipulation. In second place, some implementation details are explained. Finally, there is a summary of the results and the conclusions.

## DATA FLOW MODEL

The small file problem frequently arises in projects related to the generation of scientific raw data for later processing. Data is normally stored in Mass Storage Systems (MSS) and distributed for processing, typically through a grid platform. These projects follow a simple data flow, such as that displayed in figure 1.


Figure 1: Data Flow for Scientific Data Processing

The producer is the agent that generates the data to be processed, typically raw data obtained from measurements made by specialised devices. The MSS acts as an intermediate agent for data storage, which in the context of

* {esthera,delfino}@pic.es
† emilio@usb.ve
‡ pacheco@ifae.es

Grids could be a set of *storage elements*, which use the GridFTP [7] protocol, probably being each site a front-end of one or more MSS devices such as tape robots. The consumer is the agent that processes the data, for instance, applications that make simulations or search for data patterns. Frequently, these applications generate other files which, in turn, are stored back in the MSS. Hence, a consumer may itself be a producer.

The data is processed asynchronously, usually in a batch fashion. The reading and writing processes can be completely separated because of the WORM data access model (Write Once-Read Many). This is important because it simplifies the design; for instance, the cache coherence problem is not an important issue.

We propose a mechanism for grouping files into volumes before being migrated to an MSS. Inversely, when a file is requested, the volume is read and the file is accessed from the mounted volume transparently and handed over to the client. This mechanism is transparently placed between the file server and the MSS, and works with any kind of file service, such as FTP, GridFTP, Web Server, etc. This scheme is particularly suitable for those environments in which large amounts of data files are handled and the usage model is basically WORM, with the data flow similar to that described in the next section.

The general data flow scheme, described in figure 1, is completed with additional components that introduce the volume generation and manipulation. The data flow derived from this new scenario is described in figure 2

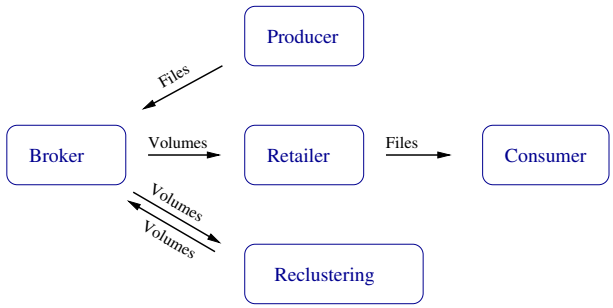


Figure 2: Data Flow for Scientific Data Processing in ViVo

The producer generates the data with a logical file name (LFN) structure. The applications (consumers) refer to data files using the same logical file names. The data is stored in brokers, which typically manage MSS, in the form of volumes. The brokers are supposed to store large amounts of data, in the order of terabytes or petabytes. The physical volumes are created in the broker side, along the file transfer process, but they could eventually be created by the producer, with the support of the broker. When the data is going to be processed, the consumers access the files through a retailer.

Files are transparently stored in volumes and the MSS do not know the internal structure of such volumes. These volumes are virtual in the sense that they do not have any locality attribute and hence they can have several physical instances or replicas. The transfer unit between a broker and a retailer is always a volume, the transfer unit between a retailer and a client is always a file. Files have a physical name that may contain information about the virtual volume in which they reside. Files also have a logical name, associated to the logical structure and it is the only name the applications know.

The volume size is defined by the broker, according to efficiency criteria. Data files can be regrouped into different volume sets, according to the access pattern exhibited by the applications, in order to maximise locality of reference. This scheme is intended to be efficient for data reading, while penalising data writing.

The retailer could be located at the broker's side, at the client's side or in an independent location. The retailer could also be an NFS server, in which the volumes are mounted once transferred from the broker. This could be helpful in a processor farm. In the context of Grids a retailer could be a storage element and the clients could execute a GridFTP from the retailer.

The reclustering process could be implemented for incrementing the access efficiency. This could be a broker's service, however it should be executed according to applications needs.

## IMPLEMENTATION

Transparency is achieved with the help of objects we call VMU (ViVo Mapping Units). These objects can be installed either in the retailer, for triggering a volume transfer, or in the client, for triggering a file transfer. It could also be installed in both systems. The VMUs are used for executing data transfers on demand. A retailer VMU would contain:

- **Component 1**: A data structure for mapping User-Provided File Names (UserFN) to a pair (Volume, File) in the broker. The UserFN is assigned by the data producer during the data upload. The pair (Volume, File) could be obtained from an internal file name, that we call a ViVo File Name (ViVoFN). This association may be implemented with the help of directory systems such as LDAP or RLS (the Replica Location Service) in case Vivo is implemented on a Grid, or simply as a file directory with symbolic link associations. For very regular file structures, the association can be implemented via lexical rules.

- **Component 2**: A procedure for managing the cache, which is invoked every time the data service tries to access a file. This procedure performs the necessary operations for mapping the UserFN to a ViVoFN, transferring the volumes and extracting the files. Alternatively, the transferred volumes can be mounted, avoiding the time-consuming process of extracting the files. This component can be implemented with the help of an **automounter** or with the help of a Virtual File System such as FUSE [8].

It is important to emphasise that an automounter needs the files to be located under a directory used as the mount point. In order to transparently trigger the script that transfers the volume, the path describing the ViVoFN should consist of a prefix that corresponds to the mount point and a suffix that corresponds to the rest of the path describing the file. For instance, the ViVoFN could be "/ViVo/MyProject/Volume0001/File0027", being the mount point "/ViVo/MyProject/Volume001" and the filename "File00027". The UserFN, could be completely different, for instance, it could be something similar to "/ViVo/MyProject/DataSet003/Measurement005.txt" defined as a symbolic link pointing to the ViVoFN. If we use a Virtual File System there are more options for establishing the association between the UserFN and the ViVoFN. In this case we could use directory systems such as LDAP.

In our implementation we have chosen the automounter option because it is a mature technology, used in many computer installations. An automounter such as amd [9] can be used for the definition of file systems, which are mountable through a user-defined script.

### *I/O procedures*

In this section we describe the basic I/O procedures that a retailer needs to do in order to be an intermediate agent between the data repository and the clients or applications.

**Reading procedure** The *Component 1* is the data structure that maps UserFN to ViVoFN. The simplest way this map can be implemented in Unix is a directory in which each UserFN is a symbolic link to its correspondent ViVoFN (initially inexistent). The access to the UserFN will be processed as an access to its correspondent ViVoFN, which in turn will trigger the automounter. The automounter script (*component 2*) will copy the volume from the retailer and mount it. Continuing with the same example, the volume could be an ISO file named "/ViVo/MyProject/Volume001.iso". This volume, once transferred, could be mounted in "/Vivo/MyProject/Volume001". When the automounter returns, the access to "/ViVo/MyProject/DataSet003/Measurement005.txt" will succeed, because it is a symbolic link to a file in the volume just mounted. The retailer can then transfer the file to the remote application.

**Writing procedure** Volume generation can be automated either at the broker or at the producer side. Let us suppose the volume creation takes place in the broker, as the producer sends the files. The producer could choose the filenames, provided the name space is organised in such a way that conflicts can not happen. For instance, the broker could assign different prefixes to every client, i.e. prefixes such as "/ViVo/MyProject", which represents an "Area" in which the client can store the files. Only after the creation of a volume in the broker, the files contained in that volume are available for reading, through an associated retailer.

**Data update** As mentioned above, the writing and updating scheme is designed for WORM systems, which is often suitable for scientific data. The file delete operation could simply be be implemented by deleting the ViVoFN reference, although the real data would be kept in the ISO volume. If the data are written on non-rewritable media, such as CDs or DVDs, the file cannot be physically removed anyway. If the file needs to be replaced, then it is added to the next ISO volume and the ViVoFN is changed for pointing to the new file, leaving the old version of the file where it originally was. A reclustering process will physically delete the file.

**Reclustering** Locality of reference, that is, consecutive access to files in the same volume, can dramatically improve the average file access time. This means that the files could be regrouped in different volumes according to the access pattern. In fact, multiple copies of the data may coexist for different applications, in the form of different volume sets. A reclustering process reads the volumes previously created and, based on new distribution criteria, rewrite the files on different volumes. The new volumes can replace the original volumes or be stored additionally in the MSS.

**Volume manipulation** It is important to be able to transfer the volumes as independent objects, not only the files individually. This is important, for instance, in data grids, where the replicas are commonly used for improving performance. In this case, of course, a high level directory must be implemented and this directory should know about the existence of the replicas.

## RESULTS

We have successfully deployed a combination of common O/S tools (mkisofs and amd) in order to handle large numbers of small files in containers, which are large ISO 9660 files. These ISO files are handled through PIC's Castor MSS. This scheme is currently in production for the MAGIC Telescope Project [10] and a Medical Image Grid project in collaboration with Parc Tauli Hospital [11]. About 15 TB have been stored in some 3500 ISO volume files.

We have observed that this scheme improves considerably the efficiency of the MSS if the file access pattern produces a high hit rate per volume. The retrieval of about 20GB of data divided into small files (around 200KB each) could take in our MSS more than a day, even if a tape drive is dedicated to the query. The same operation could take a few minutes if these files are packed in a single ISO volume, because only one tape access is needed. There exists, however, a worst-case scenario in which the file access pattern produces many volume retrievals, a condition known

as cache thrashing. This may oblige to recluster files in volumes, according to the file access pattern exhibited by the applications. Whereas a totally general implementation of Virtual Volumes would require quite complex coding, the prototypes presented are optimised for the WORM environment often found in scientific data applications.

## CONCLUSIONS AND FUTURE WORK

Research has been undertaken to deal with the small file problem issue at the data centre level. An implementation of a mechanism termed "Virtual Volumes" has been successfully tested and is in production for two research projects. This mechanism combines standard operating system tools such as symbolic links and auto-mounters together with techniques to represent a volume as a file such as the ISO 9660 specification. This solution is only suitable for WORM (Write Once, Read Many) environments, which are common in scientific data processing, because the raw data come from experiments and can not be modified.

We currently aim at achieving a better integration of this mechanism with file catalogs used by the projects we support, as well as monitoring the access patterns for developing re-clustering techniques of files in volumes.

## ACKNOWLEDGMENTS

The Port d'Informacio Científica (PIC) is maintained through a collaboration agreement of DURSI (Generalitat de Catalunya), CIEMAT (Ministerio de Educacion y Ciencia), Universitat Autònoma de Barcelona and IFAE. This research was supported in part by the Enabling Grids for EsciencE project funded by the European Union under contract number INFSO 508833, and by grant FPA2003-00417 of the Ministerio de Educación y Ciencia, Spain.

## REFERENCES

[1] Bernd Panzer-Steindel. Tape storage issues. Report to the LHC Computing Grid Project Grid Deployment Board, January 2005.

[2] http://castor.web.cern.ch/castor/.

[3] http://www-hppc.fnal.gov/enstore/.

[4] http://www.hpss-collaboration.org/.

[5] http://www.sdsc.edu/srb/.

[6] http://grid-deployment.web.cern.ch/grid-deployment/gis/GFAL/gfal.3.html.

[7] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke. Data management and transfer in highperformance computational grid environments. *Parallel Computing*, 2001.

[8] http://fuse.sourceforge.net/.

[9] Erez Zadok. *Linux NFS and Automounter Administration*. Sybex, 2001. ISBN: 0-7821-2739-8.

[10] Cortina, J. et al. Technical performance of the MAGIC telescope. In *Proceedings 29th Int. Cosmic Ray Conf. (Pune)*, pages 101–106, 2005.

[11] Carles Rubies, Josep Fernandez-Bayo, Manuel Delfino, Emilio Hernandez, Joan Guanyabens, and Andreu Pacheco. Real medical images for scientific purposes in a grid environment. In P. Inchingolo and R. Pocci-Mucelli, editors, *Proceedings of EuroPACS-MIR 2004*, 2004.