

SAND 98-1618
Unlimited Release
Printed August 1998

Effective Use of SMSS: A Simple Strategy and Sample Implementation

David Hensinger
Engineering Sciences Center
Thermal Sciences

Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1010

Abstract

The purpose of this document is to present a strategy for effectively using SMSS (Scaleable Mass Storage System) and to distribute a simple implementation of this strategy. This work was done as a stopgap measure to allow an analyst to use the storage power of SMSS in the absence of a more user friendly interface. The features and functionality discussed in this document represent a minimum set of capabilities to allow a useful archiving interface functionality. The implementation presented is the most basic possible and would benefit significantly from an organized support and documentation effort.

Acknowledgments

Thanks to Reyna Haynes lending me her expertise on SMSS.

Contents

Introduction	1
Problem Statement	1
Weaknesses of the FTP Interface to SMSS	1
Functionality Needed in the SMSS Interface	2
Sample Utility Implementation, ftp_save	3
The Functionality of ftp_save.....	3
An Example Session Using ftp_save.....	3
The Future of ftp_save.....	4
Summary	4
Appendix A: Listing of ftp_save.....	6
Distribution	8

Introduction

This report documents a simple strategy for effectively using SMSS. It also supplies a utility called **ftp_save**, which implements this strategy for single file storage and retrieval.

Problem Statement

Weaknesses of the FTP Interface to SMSS

The SMSS (Scaleable Mass Storage System) is an automated tape based archiving system designed for the long term storage of the terabytes of output produced by large scale super-computer simulations. It is available through an FTP interface from almost every classified or unclassified networked file system at Sandia. A file is stored to SMSS by initiating an FTP session with SMSS, making directories and changing directories to create a storage location, and then putting the file. A file is restored from SMSS by connecting via FTP, changing directories to locate the file and then 'getting' the file. Data stored in SMSS resides on tapes that are mounted by a robot. The tape retrieval and mounting process adds significant latency to the storage and retrieval process

The FTP interface to SMSS is clumsy for several reasons:

1. On a UNIX file system, much of the information describing the contents of a file is carried in the path of the file. For example a file located in */home/joe_user/overheat_project/simulation/static/* will probably deal with the results of a static simulation relevant to a project called overheat. In order to preserve this information in SMSS the entire directory structure needs to be recreated on SMSS before a file is stored. This is currently a tedious process.
2. Because the data stored to SMSS is often the result of expensive and time-consuming computations, there should be provisions to prevent the overwriting of data. Currently, data can be overwritten and deleted in SMSS simply by putting a file of the same name in the same location on SMSS.
3. The FTP interface does not quickly allow listing of a directory within SMSS. This makes it difficult to know what has been saved to SMSS from within a file-based directory structure.
4. An additional weakness of SMSS, which is not directly related to the FTP interface, is that the connection to SMSS goes under different names on different systems. The variety in the naming of the SMSS connection is necessary to allow different hardware connections to SMSS from the same file system; however, the provision of at least one domain name that resolves to an SMSS session on each system would greatly simplify SMSS access for users who routinely work on multiple file systems.

One way to provide this may be an environment variable set in a **system.cshrc** file defining the variable `DEFAULT_SMSS` to the most reliable domain name for the SMSS connection from that system.

Functionality Needed in the SMSS Interface

The interface to SMSS should satisfy several minimum requirements described here. These requirements are satisfied by the sample implementation of **ftp_save**. **Ftp_save** creates a pointer file (an executable script) that saves and retrieves files. It does not itself directly perform the saving and retrieving function.

When a file is saved to SMSS a pointer to that file should remain in the directory on the disk-based file system from which the file was sent to SMSS. This provides a constant reminder to the user of what has been stored on SMSS from that directory. Without this pointer a user must connect to SMSS and **cd** down the directory structure to list the contents of a directory. The latency involved in SMSS access prevents rapid queries.

The pointer remaining in the directory of origin should include all the information required to retrieve the file from SMSS. If the pointer includes all of this information, then the retrieval process and storage process can be automated, and it is much less likely the file's location on SMSS will be forgotten by the user.

As long as all of the information required to store or retrieve a file is in the pointer to the file, then it should be possible to copy the pointer to another directory, tar it into an archive, or even move it to SMSS without its becoming invalid.

The pointer should have a unique name based on the name of the file saved to SMSS. This prevents overwriting pointers with new pointers as more files are saved to SMSS. This becomes useful when numerous data files are generated in the same directory and then stored on SMSS.

When a file is saved to SMSS its it should automatically be saved with the same relative path from */home* or */* as it had on the directory system of origin. This duplicates the contextual information which helped to define the contents of the original file. An alternative to replicating directory structures on SMSS is to add the directory structure information to the file name.

A file saved to SMSS should be saved with a unique name on SMSS to prevent overwriting data. This does not mean that the file needs a unique name before it is saved to SMSS or that it must be retrieved with the unique name it bears on SMSS. Unique name extensions such as the day, date, and time are preferred over unique garbage strings.

Sample Utility Implementation, ftp_save

The sample implementation discussed here satisfies all of the requirements listed above for the archiving and retrieval of a single file to and from SMSS. The name of this utility is **ftp_save**.

The Functionality of ftp_save

The process of using **ftp_save** as a front end to SMSS begins with the invocation of **ftp_save** with the file to be saved as an argument. **Ftp_save** creates an executable script file that has several functions: it will automatically store the target file on SMSS, it will automatically retrieve the target file from SMSS, and it serves as a movable pointer to the file stored on SMSS

The file created by **ftp_save** is given a unique name based on the name of the original target file. This name will also be the name of the file when it is stored on SMSS. The name consists of the original file name with the time, day of the week, month, and calendar date and **.image** postpended. When the pointer file is invoked, it looks to see if the target file is in the present working directory. If the file is in the present working directory, it connects to SMSS via FTP, replicates the present working directory on SMSS, and saves the target file to SMSS under its own unique name. If the file is not in the present working directory, it connects to SMSS via FTP and changes directory to the location where the file was saved and retrieves the file into the present working directory under its original name.

After a file has been archived to SMSS using a script produced by **ftp_save**, the file can safely be deleted on the local disk based file system. The executable script remains as a pointer and automated retrieval method for the archived file. All of the information required to retrieve the file is stored inside the script and is independent of the location name of the script file. The file can be renamed or moved and it will still recover the stored file from SMSS provided it can connect to SMSS. Invoking **ftp_save** does not store or retrieve the file. Invoking the script which is the pointer file does that.

Part of the reliability of the strategy used by **ftp_save** is due to the way it creates executable scripts to do all of the work. Any modifications to **ftp_save** have no impact on pointer files previously created with earlier versions of **ftp_save**. **Ftp_save** does not even need to be available for the pointer files to function.

An Example Session Using ftp_save

A sample session using **ftp_save** to store and retrieve a file named **huge_file** to and from SMSS would look like this:

```
%ls  
huge_file
```

```

%pwd
/home/joe_user/working
%ftp_save huge_file
%ls
huge_file             huge_file.1503Thu_May2198.image
% huge_file.1503Thu_May2198.image
Name (smssl-atm:joe_user):Passive mode off.
Verbose mode on.
257 MKD command successful
250 CWD command successful.
257 MKD command successful.
250 CWD command successful.
200 PORT command successful.
150 Opening BINARY mode data connection for
huge_file.1510TheMay2198.
226 transfer complete.
221 Goodbye.
%rm huge_file
%ls
huge_file.1503Thu_May2198.image
% huge_file.1503Thu_May2198.image
Name (smssl-atm:joe_user):Passive mode off.
Verbose mode on.
250 CWD command successful.
250 CWD command successful.
200 PORT command successful.
150 Opening BINARY mode data connection for
huge_file.1510TheMay2198.
226 transfer complete.
221 Goodbye.
% ls
huge_file

```

It was assumed in this session that the user has already gotten a kerberos ticket to allow access to SMSS. After this session, a copy of huge_file called huge_file.1503Thu_May2198.image remains on SMSS in the directory /home/joe_user/working

The Future of ftp_save

Ftp_save is not intended to be the foundation of a comprehensive data archiving environment, but it is intended to demonstrate that significant functionality can be provided to analysts by some relatively simple tools. Analysts need a supported and documented archive management environment that surpasses the functionality of **ftp_save**. In the absence of a supported environment, **ftp_save** is a quick and dirty tool, available now, to help analysts manage data archiving.

Summary

A strategy for effective use of SMSS was presented in the form of a set of minimum requirements for a useful interface. An implementation of a utility called **ftp_save**

satisfying these requirements was presented. **Ftp_save** automates the context based archiving and retrieval of files to and from SMSS, and it provides a pointer to stored data. It was built quickly to serve the pressing needs of analysts who required a way to manage mass storage within their problem solving environment. Although it could benefit from a substantial re-write, its simple strategy does allow it to reliably provide a significant amount of functionality.

Appendix A: Listing of ftp_save

```
#!/bin/csh
#
# ftp_save - a dumb as nails ftp script builder to save a file to smss
#
# ftp_save filename
#
# ftp_save creates a file called filename.hourminutemonthdayyear.image
# this file when executed looks for filename and if it exists ftp's
# the file to smss in the same directory location it is currently
# located in. If the file filename does not exist then it gets
# the file from smss.
#
# this allows versions of the same file to be stored on smss
# and keeps a record of the stored file as the .image file
#
#
# Feb 5 4:44 PM
# David Hensinger Department 9622 845-0961
#
#
#
set the_time=`date +%H%M%a%b%d%y`
echo $the_time
set starters=`pwd | tr "/" " "`
echo "#!/bin/csh" >! $1.$the_time.image
echo "if ( -f " $1" ) then" >> $1.$the_time.image
echo "/usr/local/bin/ftp smssl-atm << eoi" >> $1.$the_time.image

whoami >> $1.$the_time.image
echo "binary" >> $1.$the_time.image
echo "verbose on" >> $1.$the_time.image

foreach item ($starters)
  echo "mkdir "$item >> $1.$the_time.image
  echo "cd "$item >> $1.$the_time.image
end
echo "put " $1 $1.$the_time >> $1.$the_time.image
echo "quit" >> $1.$the_time.image
echo "eoi" >> $1.$the_time.image
echo "else" >> $1.$the_time.image
echo "/usr/local/bin/ftp smssl-atm << eoi" >> $1.$the_time.image
whoami >> $1.$the_time.image
echo "binary" >> $1.$the_time.image
echo "verbose on" >> $1.$the_time.image
foreach item ($starters)
  echo "mkdir "$item >> $1.$the_time.image
  echo "cd "$item >> $1.$the_time.image
end
echo "get " $1.$the_time $1 >> $1.$the_time.image
echo "quit" >> $1.$the_time.image
echo "eoi" >> $1.$the_time.image
echo "endif" >> $1.$the_time.image

chmod +x $1.$the_time.image
```


Distribution

Internal Distribution:

1	MS 0151	G. Yonas, 9000	
1	MS 0841	P. Hommert, 9100	
1	MS 1002	P. J. Eicker, 9600	
1	MS 1010	M. E. Olson, 9622	
1	MS 1010	A. L. Ames, 9622	
1	MS 0835	T. C. Bickel, 9113	
10	MS 0835	D. M. Hensinger, 9113	
1	MS 9018	Central Technical Files, 8940-2	
2	MS 0899	Technical Library, 4916	
2	MS 0619	Review & Approval Desk, 12690	For DOE/OSTI