Storage Resource Broker Toolkit

Author	D. Sentenac
Date	06/03/08
Version	v0r1
Document No.	EGO-REP-COM-39

EGO - European Gravitational Observatory
Traversa H di Via Macerata - Santo Stefano a Macerata, -56021 Cascina, Italia.
Secretariat: Telephone (+39) 050 752 521 FAX (+39) 050 752 550

Change Record

Version	Date	Section Affected	Reason / Remarks
v0r1	06/03/08	All	First draft

Contents

1 Introduction

1.1 Purpose

This document is reporting on the study of the Storage Resource Broker toolkit (SRB) as a possible solution for accessing and transferring data to the Centre de Calcul of Lyon (CC).

1.2 Reference Documents

[1] Official SRB Web site: http://www.sdsc.edu/srb/index.php/Main_Page
[2] CC online documentation: http://cc.in2p3.fr/rubrique339.html?var_recherche=SRB?lang=fr
[3] Client binaries download page: http://cc.in2p3.fr/docenligne/281

1.3 Abbreviations and Acronyms

SRB: Storage Resource Broker
CC: Centre de Calcul de Lyon
SDSC: San Diego Supercomputing Center

1.4 Overview

In this document, the capabilities of the Storage Resource Broker toolkit (SRB) to manage the data transfer and data access is evaluated. SRB is a product developed by the San Diego Supercomputing Center (SDCS) and exists since 1997. It provides the abstraction mechanisms needed to implement data grids, digital libraries, and persistent archives for data sharing, data publication, and data preservation. Presented by the CC as a supported stable solution for data management, we studied the possibility to use it for managing the Virgo collaboration data stored on HPSS media. We tested in particular the data transfer in different conditions, using small and big files and under saturated bandwidth. We also present data access and manipulation test examples, focusing on what seems the most interesting features of SRB.

2 SRB TOOLKIT

This section describes the SRB toolkit, and the different steps to set-up an installation.

2.1 Generalities

Data transfer is a common task that can become complicated when dealing with huge amount of data. The transfer task does not represent a difficulty on its own and many classical tools like **bbftp** can achieve it in a fast and easily way. The main problem consists of insuring the integrity of the data copy and in particular prevents any data loss. The major task is therefore dominated by the effort of cross-checking the copy and the source in the most reliable way. The latter becomes especially difficult when dealing with particular type of storage media like HPSS which do not provide by itself an efficient way to manage the data unlike UNIX like disk storage systems. To overcome this problem, an important disk cache is generally

associated to the tape storage media. Data management is done via a a minimum software layer that allows to stage the data on disk. More interesting are software toolkits which make data management transparent, whatever the nature of the storage media. This is what SRB provides. SRB is proposed by the CC as a standard solution for managing data stored on HPSS media. More generally, SRB is a middleware that provides distributed clients with uniform access to diverse storage resources in a heterogeneous computing environment.

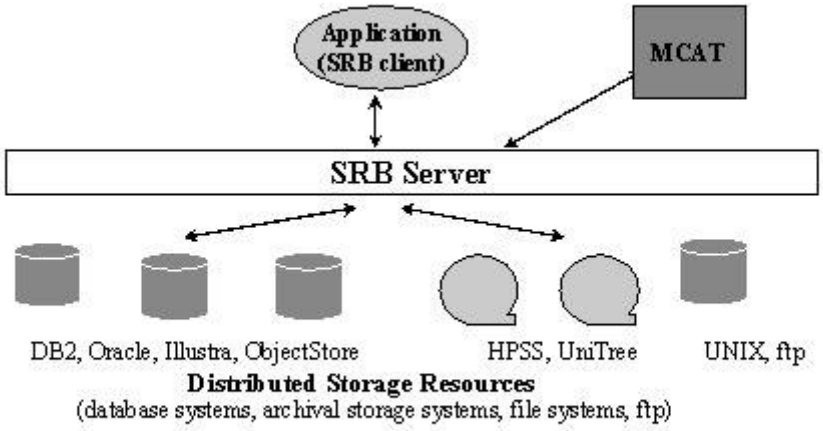

Fig 1. General view of SRB

The model in Figure 1 consists of three components: the Metadata catalog called MCAT service, SRB servers and SRB clients, connected to each other via network. The MCAT stores the Metadata associated with data sets, users and resources managed by the SRB. The MCAT server handles requests from the SRB servers. These requests include information queries as well as instructions for Metadata creation and update. Client applications are provided with a set of API for sending requests and receiving response to/from the SRB servers. The SRB server is responsible for carrying out tasks to satisfy the client requests. These tasks include interacting with the MCAT service, and performing I/O on behalf of the clients. A client uses the same common API to access every storage systems managed by the SRB. The complex tasks of interacting with various types of storage system and OS/hardware architecture are handled transparently by the SRB server.

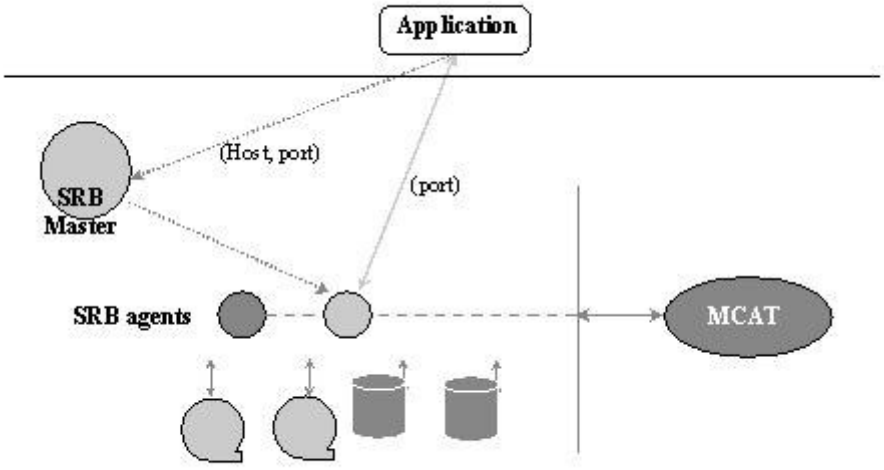

Fig 2. The SRB process model

Figure 2 depicts the SRB process model. The design of the SRB server is based on the traditional network connected client/server model. It is composed of two separate servers, SRB Master and SRB Server. The SRB Master is the main daemon listening continuously on a well-known port for connection requests from clients. Once a connection from a client is established and authenticated, it forks and execs a copy of the SRB Server, which we call SRB agent, to service the connection. From that point onward, the client and the SRB agent communicate using a different port and the SRB Master goes back to listening for more connections. A client can use the same SRB agent to service multiple requests.
Client applications communicate with the SRB agent using a set of API via TCP/IP sockets. The client library sends requests using pre-defined request stubs to the SRB agent, and receives and parses replies from the SRB agent. The model is distributed in the sense that clients and servers may be running on different hosts.
More complex architecture can be realized but are beyond the scope of this document. It consists of a group of distributed SRB servers coordinating with each other to service client requests, and forming a federation [1].

2.2 Hardware Requirements

Installation of a SRB server is proposed through the use of a Perl script by SDSC. We did not have to accomplish this task that was taken in charge by CC user-support. A SRB server v3.5.0 dedicated to the Virgo group has been installed on **ccsrb02.in2p3.fr** machine. A disk cache of 10 TB has been setup on machine **ccsrb06.in2p3.fr**. The cache size can be extended easily in the future, the politics of CC being the expansion of SRB service among experiments.

2.3 Software Requirements

The server side software is completely taken in charge by CC user-support. SRB clients can be used as simple binaries or through APIs. A good point of SRB is to provide clients for

various platforms: Linux, Suse, Solaris, Mac OS X, Windows, AIX. This is the result of the SRB philosophy which aims at providing a global and adaptable toolkit to be used in a heterogeneous computing environment converging to uniform data access.

2.3.1 Client Binaries

Client binaries are made familiar to UNIX user. They are called **Scommands**. They refer to a set of utility routines that can be used in a UNIX shell or Windows DOS command shell and access data and meta data information from SRB and MCAT. They also have a set of man pages describing each of the commands. One can type remotely command lines using **Sls**, **Scd**, **Spwd**, **Smv**, **Sput**, **Sget**, and plenty others, to manage the data. A tar file of Scommands can be downloaded at [3]. The **Scommands** were tested especially in test scripts for data transfer, as we will see in the data transfer section, and also for data management as we will see in the data access section.

2.3.2 APIs

A broad set of APIs are available in C, C++, JAVA, Perl, Python. They can be easily used altogether with Virgo software, as it will be demonstrated later in the data access section.

2.3.3 Graphical Clients

SRB offers all in one graphical clients for interfacing with data collections in addition to command line Scommands client. They include inQ, a Windows client that closely resembles the familiar Explorer interface used to organize local files, and MySRB, a web-based client.

2.4 Data path configuration

SRB collections are logical paths that point to real HPSS paths. We defined the association simply as follows at the root path for Virgo group:

HPSS : /hpss/in2p3.fr/group/virgo ←→ SRB : /ccin2p3/virgo

Therefore the following two commands are equivalent:

>Sls /ccin2p3/virgo
>rfdir /hpss/in2p3.fr/group/virgo

The notable difference when experiencing these two commands is that the “Sls” one will give for sure a result whatever the content size, while the second one may hang up in a time-out if the content is too big.

2.5 User Requirements

An account must be enabled prior to access the data. It consists of a user name a password and a domain name. Note that SRB can also use the GSI authentication layer if requested. Enabling the account is made very handy. It consists of having an environment file called **.MdasEnv** in the local home (default is ~/.srb directory) holding the connection description as follows:

mdasCollectionHome '/ccin2p3/home/virgo.cascina'
mdasDomainName 'cascina'
srbUser 'virgo'
srbHost 'ccsrb02.in2p3.fr'
srbPort '5561'
mcatZone 'ccin2p3'

Where "virgo" is the username, and "cascina" the domain name. The zone "ccin2p3" defines the domain extension of the SRB server as several domains can exist (federation of SRB servers). The fisrt time the user "virgo" logs in, he has to type the Scommand:
> *Sinit*
A password is required the first time logged in. Then, a **.srbAuthFile** is created automatically in the ~/.srb directory and no ulterior password request will be necessary afterwards.

At the administration level, file access rights can be managed exactly as being in a UNIX environment by using **Scommands** like **Schmod**. At present, a "virgo" user has been created with "read" rights permission only. It was done by simply typing:
> *Schmod –rb r virgo cascina /ccin2p3/virgo*
Where :

-r grants/changes access permissions or ownership or ACL inheritance recursively for data and sub-collections in collection and in all sub-collections under it.

–b stands for grants/changes access permissions or ownership in bulk for all data and sub-collections recursively in collection.

3 Data Transfer

Data transfer was performed using the **Scommands** client embedded in shell scripts. Several tests have been made for two main categories of data: small Ligo files (~10MB) and big Virgo files (~1GB). When transferring a file, a resource destination has to be indicated. In our case we had two possibilities according to the type of transfer we made. For big files, we used the **HPSSLyon** resource, that point to the default disk cache configuration with automatic export to HPSS. For Ligo small files we used a custom cache configuration, the **LigoDisk** resource, with no automatic export to HPSS for file merging task. Cache was configured by CC user-support.

3.1 Logs

For all transfers we tested the **Sput** and the **Srsync** command that allow uploading files and synchronizing source/remote directories, respectively. We used these commands as follows:
>*Srsync -Mvr -S LyonHPSS /archive/50Hz s:/ccin2p3/virgo/DATA/50Hz/2008*

>*Sput -vkKM –R3 -S LigoDisk /archive/ligo/v1/mdc/GHLV-HSG1_S5_R2/GHLV-HSG1_S5_R2-876998814-1000.gwf /ccin2p3/virgo/DATA/ligo/mdc/GHLV-HSG1_S5_R2*
Where:

-v stands for verbose

-K stands for checksum calculation at source location
-k stands for checksum calculation at copy location
-M stands for enabling multithreaded transfer
-r stands for recursive
-R stands for ulterior retries, here set to 3 retries if transfer failed

Other options were not thought useful to be considered here but can be consulted in [2].
These commands deliver a complete report of the final status of each transfer indifferently as follows:

->SRB:/archive/50Hz/V-885830400-31-Jan-2008-17h00-720F.50 | LOCAL:/archive/50Hz/V-885830400-31-Jan-2008-17h00-720F.50->SRB:V-885830400-31-Jan-2008-17h00-720F.50&REG_CHKSUM=60104 | 962.153 MB | 6.443 MB/s | 393.84 s | 2008.03.04 14:00:38 | 6 thr | chksum=60104

In addition to the source and copy checksums, the file size and the average speed of the transfer are indicated. From this log it is possible to ascertain that the transfer was successful. However it may happen that the logs are absent, if any kind of problem occurred (server/network connection crash or time-out etc...). This is why it is necessary to produce an additional check after a bunch of file has been transferred. SRB offers the use of **Srsync** command with the –l option that performs a file source/copy checksum and size comparison without actually doing the synchronization. An alternative less time-cost consuming (but less secure), is to make a diff on source and remote file lists, the latter being simply obtained by using a remote **Sls** command, similarly to the UNIX "ls" command. With the combination of both logs, we are pretty sure of the integrity of the transfer.

3.2 Big data files transfer

We performed tests on big files for 50Hz and trend data. Each file is about 1 GB. We had access to a backup copy in a local mount directory called **/archive/50Hz** and **/archive/trenddata**, respectively. This transfer operation is normally a routine operation, and it is interesting to automate it as much as possible. We first made manual transfers using **Sput** command, and later on propose an automate script making use of the more adapted **Srsync** command. The latter can be used to synchronize directories, similarly to the UNIX "rsync" one, and is a suitable candidate to incorporate it in a background automation script. Here is an example of such an automate shell script :

```
#!/bin/sh

while [ 1 -eq 1 ]
do
  ls /archive/50Hz > list
  Srsync -Mvr  /archive/50Hz s:/ccin2p3/virgo/DATA/50Hz/2008
  Srsync -rl /archive/50Hz s:/ccin2p3/virgo/DATA/50Hz/2008 > report.rsync
  sed -e 's%^%rm -f /archive/50Hz/%' list > rem_list.sh
```

```
  chmod +x rem_list.sh
  ls -l report.rsync | awk '{if ($5 == 0) print "./rem_list.sh" | "/bin/sh"}'
 sleep 10000
done
```

In a infinite loop, we save a local list of the files in /archive/50Hz, than perform a recursive "Srsync" on **/archive/50Hz**, than perform a "Srsync –l" to check the integrity of the transfer. If successful, we remove the source files given by the list by executing an on-the-fly created rem_list.sh script from the given list. If not successful, the files are not deleted. Then we wait more than two hours (the time necessary to receive new files to be transferred), and starts again the sequence. The file transfer log is obtained by launching the script as follows:
> Srsync_50Hz.sh > & Srsync_50Hz.log &

3.3 Small data files transfer

Small files introduce in our test an additional requirement. They need to be packed in bigger files before storage on HPSS. Indeed it is not reasonable to write small files directly onto HPSS media due to access time overload. *This is an official recommendation from the CC user-support*. Moreover, having big files improves the data reading speed and as a result, limiting the number of File I/O is a major concern for data analysis on large amount of files.
For very small data files (< 1Mb), SRB offers the possibility to pack them transparently to the user in HPSS using the bulk load option. However, this option was not used here since we decided to adopt the more appropriated custom packaging solution which is the Virgo **FrCopy** command. This was made possible in a very convenient manner, by using the **Spcommand** command client. This command allows encapsulating whatever custom command. A **FrCopy** binary was compiled on the devoted cache server **ccsrb06.in2p3.fr** (Solaris), and a Spcommand was added in the transfer sequence in the script. The sequence of transfer resulted in:

1) Transfer a bunch of small files using Sput or Srsync (in the LigoDisk cache disk resource)
2) Check integrity of the set using Srsync –l
3) If success, perform a Spcommand to merge in HPSS resource.

The **Spcommand** line reads:

Spcommand -c -H ccsrb06.in2p3.fr \"mergeLigoFiles -cin /ccin2p3/virgo/DATA/ligo/mdc/GHLV-HSG1_S5_R2 -cout /ccin2p3/virgo/DATA/ligo/mdc/GHLV-HSG1_S5_R2/pack -i "GHLV-HSG1_S5_R2-8769" -o GHLV-HSG1_S5_R2-876900814-100000.gwf\""*
Where :
-"ccsrb06.in2p3.fr" stands for the machine address of the cache server
-"mergeLigoFiles" stands for the calling script (that holds the **FrCopy** command),
-"/ccin2p3/virgo/DATA/ligo/mdc/GHLV-HSG1_S5_R2" stands for the small files resource directory
-"/ccin2p3/virgo/DATA/ligo/mdc/GHLV-HSG1_S5_R2/pack" stands for the final destination directory
-"GHLV-HSG1_S5_R2-8769*" stands for the file list (using wild cards)

-"GHLV-HSG1_S5_R2-876900814-100000.gwf" stands for the final packed file name.
This operation was implemented with the help of the user-support, who took care of the "mergeLigoFiles" script implementation and its installation. This script contains mainly the directive to export the final file onto HPSS, excluding the small files would not be kept after transfer completion.
In this way we were able to automate the transfer of Ligo files in a sequence of transfer/merging operations. Note that the same can be applied to making the **ffl** files by encapsulating the **FrDump** command in a transfer sequence.

3.4 Transfer performances

The speed of the data transfer is closely dependent of the level of cross-checks that is included in the sequence. However we observed a maximum average speed of about **7.5 MB/s** for big files (with 6 threads), while a maximum of **4 .5MB/s** for small files (with 4 threads). We interpret this difference due to the difference of thread used that was auto-managed by SRB. When using double checksum, the overall performance is reduced significantly for big files down to about **3 MB/s**, while it remains equivalent for small files. However, for small files speed performance was significantly reduced when using the "Srsync –l" list cross-check in between mergings for small files. At the end we found that an average speed of about **3MB/s** could be achieve for small or big files indifferently, using the maximum cross-check sequence (double checksum + "Srsync –l"). Note that for small files we have to consider also the file merging action in the sequence, which results not to influence significantly the average speed.

Concerning the robustness of the Scommands, we did not observe major problem in general, except when the transfer occurred in the context of a saturated bandwidth. In this stressed situation, we observed a certain amount of core-dump for the Sput command, which had no consequences on data loss. We observed generally failures leading to the following logs:

```
connectPort() --  Connection Error from server: status=-2007
connectSvr: connectPort error. status =-2007
Srsync: Connection to srbMaster failed.
Srsync:
STATUS_BAD_PACKET:
```

Or :

```
connectPort(): Connect to srbServer: ccsrb02.in2p3.fr:20269 failed, errno=111
connectSvr: connectPort error. status =-1109
Srsync: Connection to srbMaster failed.
Srsync:
CLI_ERR_SOCK_CONN: socket connect error
```

Or:

```
connectPort() --  couldn't recv status1 packet: errno=104
Connection reset by peer
```

connectSvr: connectPort error. status =-1107

They all concerned connection issues with the SRB server. The connection might have been weaken by the saturation of the bandwidth. In case of failure as mentioned above, the concerned operation was performed again. We found other situations where the Sput command overcame a failure by itself when using the retry option.

4 Data Access

In this section we focus on the data management tools of SRB, and the use of the SRB API altogether with Virgo software. It should be noted that, similarly to Xrootd, SRB has its own disk cache that allows efficient data I/O. Recently, data analysis users asked CC user-support to pre-stage on XRootD cache the full VSR1 raw data to allow fast disk access. The same can be done with SRB cache.

4.1 Use of SRB SCommands

One of the key advantages of using SRB in HPSS environment is to make it as transparent as a UNIX platform. To illustrate the difficulty of not having such a layer, we can point the crucial problem of making a "ffl" file list out of a HPSS directory using the rfdir command. If the directory holds too many files, a time-out will occur, and user-support help will become mandatory. With SRB, there are no more limitations to navigate through heavy directories, make simple operations like reorganizing directories, moving files etc…We just navigate through the catalog using UNIX like commands. A simple **Sls** list the content of a directory whatever its size, and making a "ffl" becomes easy as being at Cascina. This aspect is very important in particular to cross-check the integrity of the transferred data. All the command operations can be done remotely, so one is not obliged to log on the CC machines. This is another key point to using SRB to manage data.

4.2 Use of the SRB API

The SRB API offers the equivalent to build the Scommand binaries. One can imagine to write a fully automated software for transfers for instance. This can be done in many different languages, including C, C++, Java, Python or Perl. In addition, the API provides file I/O operations, similarly to the UNIX open, read, write, seek operations. The **FrameLib** library, or Fr package is the base library used by data analysis users. We evaluated a variation of the **FrameLib** based on the SRB API, and tested it on some key Virgo software. A **Srb** package has been created in /virgoDev/Srb/v0r1 which contains all the SRB APIs and binaries, and the **cmt** object link paths. It is the package to link on when compiling new software to work with SRB.

4.2.1 Fr

A modified version of the Fr package v20r0 has been created for tests. Fr is linked to Srb package v0r1. We tested the Fr binaries FrDump, FrCopy, FrCheck. FrDump has been used to make ffl files of the test transfers, in a remote way. As a result, with SRB, the creation of ffl at CC has became as simple as doing it at Cascina. One can find all the ffls created in the $GROUP_DIR/BKDB directory at CC. Two types of ffl correspond to the same data. The

default one pointing to the HPSS real resource **/hpss/in2p3.fr/group/virgo** directory and the SRB one pointing to the simplified SRB logical path **/ccin2p3/virgo**. The sub-directories coincide between the HPSS resource and SRB. So making a default ffl from the SRB one is direct. FrCopy and FrCheck were tested remotely as well with success.

4.2.2 DataDisplay

A version of the Dy package v20r0 has been created for tests. It is a slightly modified version of datadisplay v9r11p5, compiled with Fr v20r0. The dataDisplay was tested remotely with a local SRB "ffl". No use of Cm/dataSender is necessary in this case, and the access is not IP domain restricted by CC as is the custom Cm/dataSender. The visualization of data was tested from Cascina and the US by Marie-Anne Bizouard with success. No crash of the application was observed, and the SRB I/O seems to work very well. We noted however that the data flow is twice slower than using the Cm/dataSender method. Moreover, extra load time occurs when data files are staged from HPSS. This is normal for a long term storage system, and this drawback may be overcame by adopting a cache strategy, deciding the quantity of cache available, and by performing pre-staging operation of big data sets.

4.2.3 Vega

It was noted by Nicolas Leroy that Vega was not working well using XRootd as an interface to HPSS repository. As SRB could offer an alternative, we decided to compile a Vega SRB version on 32 and 64 bit machines. Vega then showed no more I/O problems with data access.

4.2.4 Data book-keeping

SRB can behave as a proxy for other relational databases, which provides an elegant way to enrich SRB software for data analysis. We tested the use of this feature and give an example how to filter science segment in a code application, by querying the MysQL book-keeping developed by Leone Bosi. The example can be found in the Fr v20r0 package. To make it possible, we simply created what is called a SRB "hidden object" pointing to the external database address. The latter is then accessible as would be a common file. Instead of the file name, a statement is put that contains a query to the relational database. This can be achieved using the SRB commands **Scat** or the SRB API function **srbObjOpen** inside the code.
For example the following command line:
>Scat
"virgoMySQL&SHADOW=<TEMPLATETYPE>XMLREL</TEMPLATETYPE>select timestart,timestop-timestart as elapsedtime from SEGMENTS where id_session=(select id_session from REGISTRY_SESSIONS where classname LIKE '%SCIENCE%' and isstable=1) and timestart > 863658170 and timestop < 863793073"
Where :
-virgoMySQL is the name of the hidden object
-<TEMPLATETYPE>XMLREL</TEMPLATETYPE> is the desired output format

Which gives the following XML output :

<?xml version="1.0"?>
<TABLE_RESULT>
<TABLE_NAME>
segments
</TABLE_NAME>
<TABLE_DATA>
<ROW ROWNUM="1">
<COLUMN COLNAME="timestart" COLTYPE="SMALLINT" MAXSIZE="22" PRECISION="31" SCALE="0">863671684</COLUMN>
<COLUMN COLNAME="elapsedtime" COLTYPE="SMALLINT" MAXSIZE="23" PRECISION="31" SCALE="0">7032</COLUMN>
</ROW>
<ROW ROWNUM="2">
<COLUMN COLNAME="timestart" COLTYPE="SMALLINT" MAXSIZE="22" PRECISION="31" SCALE="0">863680497</COLUMN>
<COLUMN COLNAME="elapsedtime" COLTYPE="SMALLINT" MAXSIZE="23" PRECISION="31" SCALE="0">21627</COLUMN>
</ROW>
<ROW ROWNUM="3">
<COLUMN COLNAME="timestart" COLTYPE="SMALLINT" MAXSIZE="22" PRECISION="31" SCALE="0">863702298</COLUMN>
<COLUMN COLNAME="elapsedtime" COLTYPE="SMALLINT" MAXSIZE="23" PRECISION="31" SCALE="0">42572</COLUMN>
</ROW>
<ROW ROWNUM="4">
<COLUMN COLNAME="timestart" COLTYPE="SMALLINT" MAXSIZE="22" PRECISION="31" SCALE="0">863745155</COLUMN>
<COLUMN COLNAME="elapsedtime" COLTYPE="SMALLINT" MAXSIZE="23" PRECISION="31" SCALE="0">45576</COLUMN>
</ROW>
<ROW ROWNUM="5">
<COLUMN COLNAME="timestart" COLTYPE="SMALLINT" MAXSIZE="22" PRECISION="31" SCALE="0">863790875</COLUMN>
<COLUMN COLNAME="elapsedtime" COLTYPE="SMALLINT" MAXSIZE="23" PRECISION="31" SCALE="0">1960</COLUMN>
</ROW>
</TABLE_DATA>
</TABLE_RESULT>

The same can be achieved inside a code using the function srbObjOpen :

srbObjOpen
(conn,"virgoMySQL&SHADOW=<TEMPLATETYPE>XMLREL</TEMPLATETYPE>select timestart,timestop-timestart as elapsedtime from SEGMENTS where id_session=(select id_session from REGISTRY_SESSIONS where classname LIKE \"%SCIENCE%\" and

isstable=1) and timestart>863658170 and timestop<863793073", O_RDONLY, "/ccin2p3/home/virgdata.cascina")
Where:
-"/ccin2p3/home/virgdata.cascina" is the path location of the hidden object.

Note that access rights can be managed on the hidden objects as for a file. With this technique it becomes possible to enrich easily the SRB **FrameLib** with filters functions.

5 Conclusion

We have investigated the properties and performances of the SRB toolkit, by implementing several data transfer tests, and data access tests. We appreciated the easiness to work into the SRB environment, by setting up the ".MdasEnv" file to get started, by using the highly accessible set of APIs and binaries clients. We found good performances for the transfers to HPSS up to about 60MBits/s without checksums that makes it comparable to bbftp. We enjoyed the plethora of UNIX like Scommands that we could easily embed in classical shell scripts to make transfers, ffl files or simply to manage remotely the data repository. The SRB API library has proven to be stable and robust, and has made realistic in a short term the use of a SRB version of the Common Virgo Software. We already can provide a set of SRB versions of a dataDisplay, Vega, and Fr binaries, like FrDump, FrCopy, FrCheck that can be used remotely. We benefit a lot from the help of CC user-support (Jean-Yves Nief for SRB and Andrei Moskalenko for HPSS), while giving feedback on the ongoing tests, in collaboration with M-A Bizouard and N. Leroy. We understood clearly that the aim of SRB is to allow sharing various storage media transparently in a friendly UNIX way, altogether with allowing remote data access, and appears to be the best solution so far to handle data in HPSS storage media. We can conclude that we are convinced that this tool is reaching its goal without problems, and is a good compromise solution to the Virgo community at CC.

___oOo___