A Lightweight, Scriptable, Web-based Frontend to the SRB

Toby O. H. White[1], Rik P. Tyer[2], Richard P. Bruin[1], Martin T. Dove[1], Katrina F. Austen[1]

[1]Department of Earth Sciences, Downing Street, University of Cambridge. CB2 3EQ
[2]CCLRC Daresbury Laboratory, Daresbury, Warrington. WA4 4AD

Abstract

The San Diego Supercomputing Centre's Storage Resource Broker (SRB) is in wide use in our project. We found that none of the supplied interfaces fulfilled our needs, so we have developed a new interface, which we call TobysSRB. It is a web-based application with a radically simplified user interface, making it easy yet powerful to operate for novices and experts alike. The web interface is easily extensible, and external viewers may be incorporated. In addition, it has been designed such that interactions with TobysSRB are easily automatable, with a stable, well-defined and well-documented HTTP API, conforming to the REST (Representational State Transfer) philosophy. The focus has been on lightweightness, usability and scriptability.

Introduction

Introduction to the SRB

The Storage Resource Broker (SRB) is a distributed data storage product written and maintained by the San Diego Supercomputing Centre (SDSC)[1]. It is in wide use throughout the UK eScience community [2].

It is intended to allow users to access files, and collections of files seamlessly within a distributed environment. It provides an abstraction layer between the storage of data, which may be in multiple locations, on multiple filesystems, and the access of data, which is presented through a unified interface, transparent to the details of storage type or location.

An SRB system consists of four components:

- The MCAT (Metadata Catalogue) database, which stores internal SRB information - most importantly, mappings between a file's SRB address, and its storage location.
- The MCAT SRB server, which contains much of the logic for manipulating files and internal SRB state.
- The SRB server, which exports an interface across the network, accepting requests from clients, and translating these requests into interactions with the MCAT database and MCAT server.
- The SRB client, which is what the user or developer sees, and interacts with, and which exports the transparent, seamless, abstraction layer which is the point of the SRB.

Within the *e*Minerals project, we have been using the SRB for several years now as our primary data repository. It enables us to share data across the project in a very simple fashion. We have also built workflow tools using the SRB as a universally-accessible storage layer[3]. These tools are widely used across the project, and several large-scale studies have been performed using them.

SRB interfaces

From the user's perspective, regardless of the interface used, the SRB appears much like a distributed filesystem, with data in files, and files in collections that may be nested like directories.

Interfaces for users

SDSC provide three user-accessible front-ends to the SRB, through which end-users may interact with files, and navigate through the collections and datasets of the SRB:

- The Scommands, which are a series of command-line tools for Unix-like operating systems, written in C, which very roughly mimic native POSIX commands; thus to list the contents of a collection, `Sls` is used, where `ls` lists directory contents on a traditional Unix file system.
- MySRB, which is a web-based graphical interface.
- InQ, which is a native graphical MS Windows application.

In addition, there are a number of third-party user interfaces available, which all support subsets of the SRB's functionality[4].

Interfaces for developers

In addition, there are a number of officially supported SRB APIs available:

- There is a fully-featured C API, which is the primary developers' interface.
- Jargon is a Java API which exports the full range of functionality.
- Matrix exports a WS (Web Services) SRB interface, which is built on top of Jargon.

Again, there are also a number of third-party interfaces. Mostly these consist of language-specific wrappers around the C API.

Use of the SRB

The SRB's primary selling point is as a way to abstract access to files stored in multiple logical locations, through a single interface, which bears a resemblance to a hierarchical filesystem. In addition, it offers a number of additional features - limited user-editable metadata, and replication of files.

However, within our project, we have found that the only aspect of the SRB that we are interested in is the common view of a familiar filesystem-like interface.

This manner of usage is encouraged by the analogies that can be drawn between the Scommands and native unix filesystem tools.

Indeed, for both SRB versions 2 and 3, there are filesystem plugins, which enable the use of the SRB transparently, and allow an SRB repository to be mounted as a native filesystem within Linux, and a similar plugin to Windows Explorer, which enables an analogous interface for Windows.

Given that we use SRB only as a network filesystem, many of the features offered by the existing interfaces are beyond our needs.

Deficiencies in current methods of SRB interaction

User interfaces

The Scommands are the SRB developers' recommended interface. However, navigating an SRB repository through the Scommands has many of the same strengths and drawbacks as navigating a normal filesystem from the command line.

In its disfavour is the fact that visualizing a directory structure, and navigating through it when you are not sure of the destination, can be clumsy from the command line; and for users unused to command-line interaction, it is a daunting prospect. Indeed it is for this reason that graphical file system browsers were invented in the 1970s.

Furthermore, a network filesystem can never deliver the performance of a local filesystem, due to network latency. This is a familiar problem, from which all network filesystems (NFS, AFS) suffer. Thus Scommand-ing one's way through a repository is always slower and less efficient even than navigating through a filesystem using the analogous native unix tools.

A further deficiency is the necessity to install the Scommands on any system where access is required. It is impossible to connect to the SRB using this method from an arbitrary computer which knows nothing about the SRB. Furthermore, even when the Scommands are installed, it is necessary for each user individually to be set up correctly to use the SRB - thus it is not possible to, for example, lean over someone's shoulder, while they are logged on, and quickly retrieve a file from your own SRB collection.

In its favour is the fact that, since the interface is expressed and used through a unix shell, SRB interaction can be very easily incorporated in a script, whether written in shell script, or in a higher-level language such as Perl or Python.

However, again, network effects conspire to make repeated **`Sls`**'s a great deal slower than lots of **`ls`**'s. In addition, our experience has been that the SRB is not sufficiently robust to allow scripting of many SRB interactions (on the order of a few hundred in less than a minute). This is due to an architectural flaw in the SRB server, which will report success for a transaction, even before the database has been updated, thus breaking the atomicity of SRB operations. Clearly this introduces an enormous number of potential race conditions. It is therefore impossible to set up the SRB infrastructure to allow high volumes of requests of the sort which are essentially trivial for a real filesystem. Repeated failures of various sorts will be seen.

MySRB and InQ both try to solve some of the problems associated with the Scommands, in different ways.

InQ primarily tries to solve the problems associated with a textual interface. As a native Windows application, it must be installed locally, on a Windows computer. Thus, it does not solve the problems associated with needing to be at a particular computer. It does at least not require setting up for each user - it can be used to retrieve anyone's files from a single installation.

MySRB is a web-based interface to the SRB. It is a CGI script, implemented in C, which provides a stateful session of limited duration, and most SRB actions are possible through it. Since it is a web application, it is accessible from anywhere with a working web-browser and internet connection.

However, it suffers from a number of deficiencies in its user interface (UI). Firstly, due to the necessity to make available through MySRB almost all of the functionality of the SRB, the interface suffers from complexity and overcrowding.

Secondly, when using it as a simple method for retrieving files, there are two particular irritations which render it frustrating for the expert, and confusing for the novice.

- It second-guesses the browser's ability to show files - if a request is made to view a file of a type MySRB is not aware of, or indeed that MySRB thinks cannot be displayed by the browser, then MySRB will escape all HTML-sensitive characters, and insert the resultant translated file into an HTML frame between **`<PRE>`** tags. This naturally renders it impossible for the browser to render the output sensibly when retrieving an SVG file.
- When downloading a file, MySRB communicates with the browser such that every time, the browser's save dialogue tries to name the file **`MySRB.cgi`**.

Developer interfaces

In terms of scriptability, the available developers APIs did not fit our needs either.

Jargon is in Java, which is of course only useful for Java applications. Very few of our tools are written in Java; and Java is not conducive to quick scripting of the sort that the Scommands can be used for.

Matrix is a web-services API, and as such is nominally platform-independent. However, in practice, it requires a large investment in the stack of SOAP/WS infrastructure on any computer which must communicate with it. Again, there is no sense in which it can be used in a script-like fashion.

Of the third-party interfaces, none suited our purposes.

Motivation

A further problem with all the available interfaces, with the obvious exceptions of MySRB and Matrix, is that they all require communication between client and server over a number of uncommon ports; primarily port 5544 for MCAT communication, and variable ports from 65000 upwards. This makes use of the SRB from behind firewalls tricky. Clearly MySRB and Matrix work entirely over ports 80 or 443, which are almost universally available.

So, in summary, the main problems we perceived with the available methods of interacting with the SRB were:

I. The UIs of the available graphical approaches were insufficiently user-friendly
II. None of the interfaces available offered sufficiently robust scriptability.
III. Only the Scommands offered any scriptability at all, but required local installation and setup for each user, and could not be used from behind many firewalls.

To this end, we have built a new SRB frontend, which we call TobysSRB, which ameliorates all of these issues.

Approaches to a solution

The primary motivation for the creation of TobysSRB was to solve problem I above - we needed a very simple UI for retrieval of files from the SRB; its initial intended audience was in fact undergraduate students, who had little to no knowledge of the software involved, and who had little to no control over the computers from which they need to access the SRB.

The following requirements were thus essential.

- It should allow the very easy retrieval and viewing of files.
- It should not require any installation on client machines.

It was clear that a web-based solution would best fulfil these criteria, since web browsers are universally available, and a small amount of forethought combined with extensive user testing and feedback would ensure that the UI could be made sufficiently transparent that novice users would feel at ease, solving problem I above.

In addition, since we also perceived problems II and III above, we realised that, with proper design, a web-based solution could fix these also.

Problem II can only sensibly be solved by wrapping the Scommands - a non-robust, but scriptable, interface - behind a layer which performs proper error and timeout checking. This layer can as well be a web-based application as any other sort.

Problem III is of course solved in the same way - by making the interface web-accessible, it is then accessible anywhere.

Overview of TobysSRB

TobysSRB is implemented as a CGI script, written in Python, and all its interactions with the SRB are performed by executing appropriate Scommands using Python's subprocess handling.

For security, it is written such that it will only work over an HTTPS connection; the only information exposed to eavesdroppers is that a connection is being made; all data is encrypted.

It requires no special configuration on the server other than that required for running CGI scripts in general; and of course that the Scommands be installed somewhere on the server.

In this section, we shall explain firstly its internal implementation, then the interfaces presented to the user, both through a web-browser, and through its web-accessible API.

Internal implementation

Configuration and authentication

Configuration of the Scommands for a user involves the creation of a directory, **`~/.srb`**, and two files therein, **`~/.MdasEnv`** and **`~/.MdasAuth`**. Whenever the user wishes to interact with the SRB, they must first issue the command **`Sinit`**. This authenticates them against the server, and then creates a file within **`~/.srb`** which keeps the SRB session's state (which consists only of the location of the current SRB collection). The session should be ended with an **`Sexit`** which merely clears up this session file.

(Of course, frequently one will pause in the middle of a session, and forget to do an **`Sexit`** before logging out, with the result that the **`~/.srb`** directory quickly fills up with stale session files.)

```
mdasCollectionHome '/home/tow.eminerals'
mdasDomainHome     'eminerals'
srbUser            'tow'
srbHost            'forth.dl.ac.uk'
srbPort            '5544'
defaultResource    'CambsLake'
AUTH_SCHEME        'ENCRYPT1'
```

Figure 1: Example **`.MdasEnv`** File

A typical example of a **.MdasEnv** file is shown in figure 1. (The **.MdasAuth** file contains nothing but a password.) Note, however, that much of the contents is either redundant, or irrelevant to the client.

In almost all cases, the home directory can be constructed from the username - and in any case, the client ought not to need to specify their home directory when the server will also know. The client ought not to care about the authentication mechanism, when the server could tell it. Since there is a default port, it should not be necessary to specify it unless using a non-default value.

Thus, in fact the only information that the client should need to know is username, password, and location of the MCAT server. In our case, since TobysSRB is wrapping all SRB details, the client need not even know this last. All the user need specify to a given TobysSRB instance are username and password. All additional information is held by TobysSRB in a configuration file.

So in a given session with TobysSRB, the username and password are provided as CGI variables. TobysSRB then constructs a temporary directory, within which it constructs two files, corresponding to **.MdasAuth** and **.MdasEnv**. All necessary Scommands are then executed as follows:

```
MdasEnvFile=$TMPDIR/MdasEnvFile   \
MdasAuthFile=$TMPDIR/MdasAuthFile \
Scommand
```

and at the end of a TobysSRB session, the files and the temporary directory are removed.

A brief note on password security: since all TobysSRB sessions occur over HTTPS, all information on passwords is secure from eavesdropping. However, it may be visible, as a CGI variable, in the URL bar of a web-browser. In order to obviate the possibility of password stealing by looking over shoulders, it is trivially obscured by firstly XORing the password character by character, and then encoding all URL-sensitive characters. This makes the password essentially immune to being picked up by a glance.

Session state

One of the main reasons why MySRB is difficult to script against is the fact that it is a stateful application, and cookie-handling is used to keep the session alive. This is illustrated in figure 2.

This has the advantage, of course, that the user is not required to reauthenticate every time they perform some action - rather the authentication data is held in a cookie. And since the authentication information required for MySRB is not merely username and password, but the full gamut of configuration information described in the previous section, it would be obnoxious to require typing in every time.

Unfortunately, of course, it also means that any client must be prepared to handle cookies to interact with MySRB, which effectively restricts clients to browsers, and excludes simple scripts. It also makes the internal workings of MySRB significantly more complicated, since session-handling logic is required.

However, TobysSRB works in an entirely stateless fashion, as shown in figure 3. This effectively means that reauthentication occurs on every action. However, this can be made transparent to the user - once the user has authenticated, every page that is returned from TobysSRB has a username/password form, but the values are filled in by default, so the user need not worry about them.

For a session consisting of multiple commands , this stateless approach theoretically involves an increase in load on the server side, since now a temporary directory and files must be created and destroyed for every request. (For single requests there is obviously no difference.) However, in practice, we have found this increase entirely unmeasurable. And from the client's perspective, any marginal increase in time is insignificant compared with the time required for each interaction between TobysSRB and the underlying SRB servers.

In addition, MySRB need only initialize the SRB session once for each MySRB session, whereas TobysSRB in principle must reinitialize every time. However, since TobysSRB is stateless, and the only purpose of **Sinit** is to set the current directory, in fact we need not initialize our session at all. If all SRB locations are specified as full (not relative) paths, then Scommands all work perfectly correctly without initialization; and this also obviates the need for us to worry about clearing up stale session files later on.

Furthermore, by keeping the implementation entirely stateless, it means that the interface is trivially scriptable, since no cookie-handling need be performed by the client.

Interaction with the SRB and error handling

As described, interaction with the SRB is performed by Scommands executed from within TobysSRB. Some notes on implementation here are worthwhile.

Firstly, since the Scommands are being executed, by the shell interpreter, it is vitally important that any input that is passed in from the user is checked before constructing the command line, otherwise malicious shell commands could be easily executed by the user. To this end, we check for safety all user input that will be passed to the command line, and escape any characters in SRB filenames that are also shell metacharacters.

This task is made more difficult by the fact that (prior to version 3.4.1 of the SRB) there is no definitive list of what characters are allowable in SRB filenames - indeed different official SRB tools allow different sets of characters, and files created through one tool may not be retrievable through another (*vide* frequent discussions on SRB-Chat); and the list of problematic characters varies between SRB versions. Therefore, we simply disallow any characters that might cause problems on any version of the SRB. This does prevent certain filenames, which are otherwise legal in recent SRB versions, from being used, but we feel our conservative approach is entirely justified.

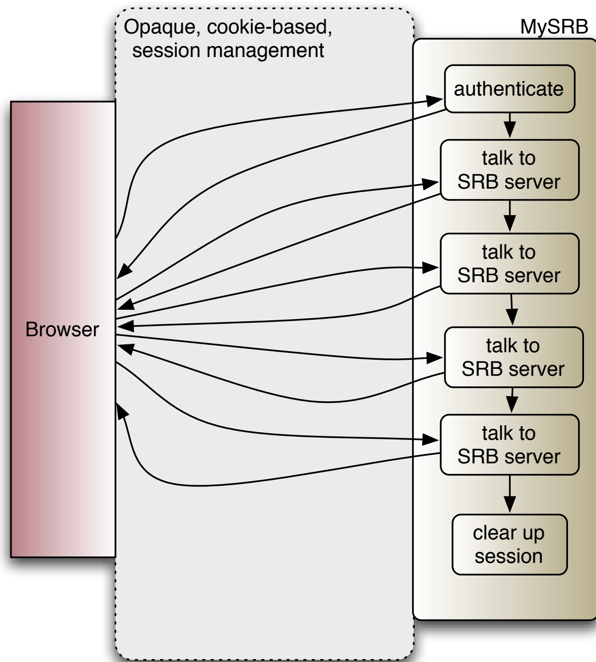

Figure 2: Illustration of MySRB session

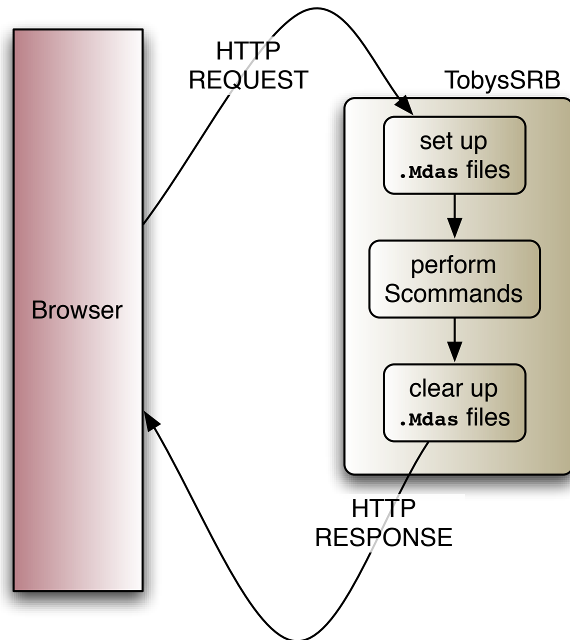

Figure 3: Illustration of TobysSRB session

Secondly, interactions with the SRB are never entirely robust. For example, occasionally an Scommand will hang indefinitely, or return with an obviously wrong error message.

This is excusable (or at least it is feasible to work with) when using the Scommands by hand - one can easily interrupt a hanging command; or re-execute one that has returned wrongly. However, when automating interactions, it is a major drawback, and thus TobysSRB is intended to wrap the Scommands and insulate the user against such vagaries.

Thus, each command is executed in a subprocess with a timeout, and TobysSRB repeatedly checks the status of each command issued; should the timeout be repeatedly exceeded, TobysSRB will return an appropriate error to the user.

Furthermore, the error handling of the Scommands is highly inconsistent - no documented scheme of error codes exists, so the cause of errors can only be deduced from reading the output of the commands; some of which are output on stdout, some on stderr; and there appears to be no pattern to their format. Furthermore, some error codes are overloaded, and the meaning of the error can only be deduced from the context of the request.

Therefore TobysSRB will also inspect both the stdout and stderr returned by the Scommands, and parse them to discover the cause of the error. Where the error appears to be of the type that is known to occur spuriously, the Scommand will be reissued a few times in the hope that it will succeed.

Finally, if the error occurs repeatedly, then TobysSRB will return to the user the error message, accompanied by an HTTP status code indicating the type of error.

As far as possible, TobysSRB will report error codes in accordance with the meanings assigned by HTTP/1.1[5]. Thus, 200 is only returned if the request was successful. If an upload was successfully performed the status is 201. If a request fails due to an authentication error, then the status is 401, while if the failure is due to an SRB timeout, the status is 408, and so on.

This enables TobysSRB to fit within the general framework of HTTP applications, and means any scripts written against it can deal with failures robustly and intelligently.

Extensibility

Because TobysSRB is written in well-constructed Python, with none of the complications associated with session management, it is a bare 500 lines long. Its control flow is thus easily grasped, and it is easily extended.

This was illustrated when a requirement arose to process XML files specially, by providing additional links to an external service which would transform the XML into a form more easily viewable in a web browser (described in further detail in [6]). It was a matter of ten extra lines of code to include this additional functionality.

Web application UI

For security, TobysSRB will only work over an HTTPS connection; if accessed over unencrypted HTTP, it will refuse to grant access, and try to redirect the browser to an appropriate HTTPS address.

On first accessing TobysSRB, the user is asked for a username and password, from which the location of the user's home collection is established, and a listing of the contents of that collection is

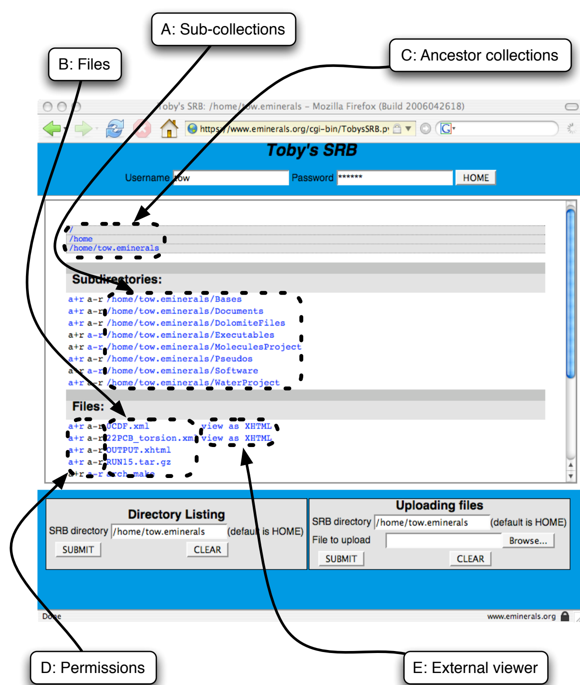

Figure 4: Screenshot of TobysSRB

retrieved and presented to the user. A screenshot is shown in figure 4.

As can be seen, collections (A) and individual data objects (B) are listed separately. The collection names are links which will return a page of the same format, displaying the contents of that collection.

The filenames are all links which will directly return the files; thus following them will allow the browser to render them however it can, and using the normal browser mechanism (usually right-click and select “Save As...”) will allow downloading the file. By constructing the URL appropriately, we have ensured that when saving the file, the browser knows the filename and will save it under that name.

In the top left of the screen, there is a list (C) of all the parent collections, which allows quickly navigating back up the SRB.

At the bottom of the screen are two forms; the first allows for quickly listing a given directory rather than having to navigate page-by-page throughout the hierarchy; the second allows uploading local files.

Two further things are worthy of note. Firstly, notice that each collection or filename is preceded by two links (D) labelled '`a+r`' and '`a-r`'. These change the permissions on the file or collection (recursively for collections) and add and remove, respectively, global read permissions.

The SRB of course offers much greater granularity of permissions, but we have found that the only changes we generally make are to global read permissions, and a quick button click is distinctly easier than `Schmod`'s arcane and poorly documented syntax. Trying to allow for the full range of allowable permission modifications would significantly complicate the interface.

Finally, as previously mentioned, all files suffixed `.xml` may have an additional link appended (E). This is because most of the XML files we produce within our project are CML files, and we have also developed an on-the-fly CML-to-XHTML convertor to allow easy viewing of such files[6].

Scriptable API

Part of the purpose in creating TobysSRB was to provide a stable and robust programmable interface to the SRB, accessible from computers where the Scommands are not installed. For this reason, all TobysSRB functions are accessible with a simple and well-documented API, described in this section.

TobysSRB receives information from a client on two channels;

- HTTP verbs
- CGI argument list.

HTTP verb

TobysSRB understands the following four HTTP verbs, and performs appropriate actions:

- GET - for retrieval of information; either of a directory listing or of file contents.
- PUT - a file will be uploaded.
- DELETE - a file or collection will be deleted.
- POST - one of a variety of other actions may be taken.

When TobysSRB is invoked from a web-browser, only GET and POST will be used; but PUT and DELETE can be used from scripts which issue HTTP requests.

CGI argument list

A range of CGI arguments are recognized, which TobysSRB will act upon. A full description of the API is beyond the scope of this paper, but by appropriate combinations of parameters, all of the SRB actions TobysSRB knows about may be performed.

The API is clearly documented, and easily understood. Most importantly, files are easily accessible from a single, stable, easily generated URL, which looks like

```
https://my.server.ac.uk/TobysSRB.py/
path/to/filename?
username=user;password=pass;
```

By accessing that URL with GET, the object may be retrieved from anywhere, accessing it with PUT will place data into the file, accessing it with DELETE will remove the file, and - in concert with additional CGI parameters - accessing it with POST will perform all other available operations.

If the URL ends in a ‘/’ then it is assumed that the relevant SRB object is a collection, and so a listing of its contents will be returned; if not then it is assumed to be a data object, whose contents will be retrieved and returned. If either assumption is wrong, a redirect will be issued, in the same way as would happen for an HTTP request.

Philosophy

This API very much follows the REST (Representational State Transfer) philosophy [7] and may be seen as a lightweight alternative to wrapping the SRB with Web Services. Each SRB collection and file may be perceived, through TobysSRB, as an HTTP-accessible resource, upon which various operations (retrieval, modification, deletion, *etc*.) may be performed.

In this fashion, all of the SRB operations supported by TobysSRB may be used from a script capable of running on any internet-connected machine, without recourse to the Scommands.

Thus the command

```
Sget /path/to/file_of_interest
```

may be replaced, when Scommands are unavailable, with

```
wget "https://my.server.ac.uk/
TobysSRB.py/path/to/file
username=user;password=pass"
```

Other interactions are easily performed by generating more complex HTTP requests, some of which can be done with **`wget`** or **`curl`** (which are almost universally available on the Unix command line), and for those which cannot, generation of an HTTP request is less than 10 lines of easily abstractable Perl or Python since modules exist for this in the standard libraries of both.

It should be noted that the SRB username and password must be known by the script in order to create the URLs. In simple cases, they could be stored inline in the script, but they could equally be dynamically read from a file, or generated by user input. This does not however make the method any less secure than other existing methods the Scommands need to know these data as well, but simply store them in the **`.srb`** directory. By allowing them to be stored or generated elsewhere, we place more power in the hands of the user, since they do not need to create **`.srb`** directories on every machine; nor are the usernames and passwords so easily discoverable should a client account be compromised.

This API therefore provides an accessible way of automating SRB interactions. In addition, since TobysSRB acts as a buffering layer against the vagaries of the SRB, and returns well-documented error messages, it can act as a considerably more robust SRB interface for scripts, even where the Scommands are available. Since no additional client software is necessary, it can be immediately used on any internet-connected machine without preparation. And finally, the interaction occurs wholly over a single port, which generally speaking will be port 443, and universally available, so the resultant scripts are portable to any client machine without the need to worry about firewall issues.

Comparison with MySRB

Clearly, in some respects, TobysSRB is a direct replacement for MySRB, as a web-based interface to the SRB. A brief comparison follows.

MySRB supports a much wider range of SRB operations. It is the primary interface where new features in the C API are prototyped and exposed to the user.

In comparison, TobysSRB supports only the operations

- file retrieval
- file upload
- file delete
- collection listing
- collection creation
- collection removal
- add/remove global read permissions

However, in our experience, these compose by far the vast majority of operations performed, and are the only ones that we have found it useful to script - in any case, all other operations remain available through the Scommands.

Because TobysSRB supports a much reduced range of operations, its UI can be much simpler. This means that it could be designed in a fashion analogous to a typical graphical filesystem browser, which makes it easily accessible to any computer user familiar with that paradigm.

In addition, MySRB makes a distinction between viewing a file and retrieving it - when viewing it, MySRB second-guesses the browsers rendering capabilities, and massages the output in various ways before rendering it in a frame. This means that, for example, an SVG file may not be viewed through MySRB, because it is transformed into fully escaped HTML and presented to the browser as text.

TobysSRB, on the other hand, presents the file straight to the browser, mime-typed accordingly, and allows the browser to render the file how it, or the user, chooses.

Further, since MySRB is a stateful application, which uses cookies for session handling, it is very complicated to automate a MySRB session.

TobysSRB, however, works entirely statelessly, and therefore scripting an upload or download is as simple as performing (through **`curl`**, or **`wget`**, or Perl or Python) an HTTP request to a URL.

Finally, although the source for MySRB is available, so in principle it could have been altered in order to fulfil our requirements, and would be extensible for the addition of external links, in practice this is not the case, since it consists of 18000 lines of code, which is somewhat opaque to the uninitiated. In contrast, TobysSRB consists of 500 lines of Python, and is much more easily altered.

User experiences

TobysSRB is now in use across the (multi-institutional) *e*Minerals project which employs the authors, both for project members, and for the undergraduate students who work with the project. In addition it has been disseminated to a number of other users within the institutions hosting *e*Minerals. To the best of the authors' knowledge, every user who has been exposed to TobysSRB prefers it to MySRB.

Some users, especially the undergraduate students at whom it was initially aimed, use TobysSRB as their only method of SRB interaction. Other more advanced users continue to use the Scommands for some, if not most tasks, but when web access is required (for use from remote computers) or a browser interface preferred (for viewing XHTML or SVG files), then TobysSRB is unanimously preferred to MySRB.

The scriptable interface has not yet been as widely adopted, largely because there are a number of existing tools which already use the Scommands as their primary interface. However, several users do prefer the TobysSRB API, and a growing number of newer scripts are being written to work with that interface.

Summary

Finding the currently available methods for interaction with the SRB to be inadequate for our needs, a new front-end was developed. This interface, TobysSRB, pares down the facilities offered by MySRB and in so doing allows for the generation of a considerably more user-friendly interface.

TobysSRB allows the viewing of any SRB file, and is easily extensible such that it can, for example, automatically create links to external services, such as the CML viewing tool described in [6]. The primary objective of TobysSRB was to provide users within the *e*Minerals project with a user- friendly interface that performs all the commonly required tasks with greater facility than those tools already available.

Furthermore, the requirement for intelligent error-detection and timeout handling when wrapping the Scommands has resulted in the creation of a *REST*ful HTTP API for interacting with the SRB, allowing SRB interaction in a robust and network-transparent, universally accessible fashion.

Both objectives have been achieved, and TobysSRB has become a transferable tool that can be used by any project using the SRB.

Acknowledgements

We are grateful for funding from NERC (grant reference numbers NER/T/S/2001/00855, NE/C515698/1 and NE/C515704/1).

References

[1] Moore, RW and Baru, C., *"Virtualization services for data grids"* in *"Grid Computing: Making the Global Infrastructure a Reality"* (ed. Berman, F. Hey, A.J.G and Fox, G., John Wiley) Chapter 11 (2003);
Also, see http://www.sdsc.edu/srb

[2] Doherty, M., *et al, "SRB in Action"*, All Hands Meeting, Nottingham, 2003;
Manandhar, A. S. *et al.* *"Deploying a distributed data storage system in the UK National Grid Service using federated SRB"*, All Hands Meeting, Nottingham, 2004;
Berrisford, P., et al., *"SRB in a Production Context"*, All Hands Meeting, Nottingham, 2004

[3] Bruin, R.P., et al. *"Job submission to grid computing environments"*, All Hands Meeting, Nottingham (2006) - in press;
Calleja, M. et al. *"Grid Tool integration with the eMinerals project"*, All Hands Meeting, Nottingham (2005);
Calleja, M. *et al*., *"Collaborative grid infrastructure for molecular simulations: the eMinerals minigrid as a prototype integrated compite and data grid"*, Mol. Simul. **31**, 303 (2005)
Chapman, C. *et al., "Managing Scientific Processes on the eMinerals mini-grid using BPEL"*, All Hands Meeting, Nottingham (2006) - in press

[4] http://www.sdsc.edu/srb/index.php/Contributed_Software

[5] Fielding, R. *et al., "Hypertext Transfer Protocol -- HTTP/1.1"*, RFC 2616, 1999.

[6] White, T.O.H. *et al*., *"Application and Use of CML in the eMinerals project"*, All Hands Meeting, Nottingham, 2006.

[5] Fielding, R. *et al., "Hypertext Transfer Protocol -- HTTP/1.1"*, RFC 2616, 1999.

[7] Fielding, R., *"Architectural Styles and the Design of Network-based Software Architectures"*, PhD thesis, University of California Irvine, 2000.