Integration of HDF5 and SRB for Object-level Data Access*

Peter X. Cao
Univ. of Illinois, Urbana
xcao@ncsa.uiuc.edu

Mike Wan
Univ. of California, San Diego
mwan@sdsc.edu

Mike Folk
Univ. of Illinois, Urbana
mfolk@ncsa.uiuc.edu

Abstract

Fast partial access to objects from very large files in the SDSC Storage Resource Broker (SRB[5]) can be extremely challenging, even when those objects are small. The HDF-SRB model integrates the SRB and NCSA Hierarchical Data Format (HDF5[6]), to create an access mechanism within the SRB that is more efficient than current methods for accessing object-based file formats.

This model integrates two successful technologies, the SDSC SRB and the NCSA HDF, to create a new, more sophisticated distributed data service. The SRB serves as standard middleware to transfer data between the server and client. HDF5 provides interactive and efficient access to datasets or subsets of datasets in large files without bringing entire files into local machines. A new set of data structures and APIs have been implemented to support such object-level data access. A working prototype of the HDF5-SRB data system has been developed and tested.

1. Introduction

Storing massive data presents two big challenges: management of distributed data systems and efficient access to complex data content. The NCSA Hierarchical Data Format (HDF) and the SDSC Storage Resource Broker (SRB) have addressed the two issues. The SRB is client-server middleware (or grid data software) that provides a uniform interface and authorization mechanism to access heterogeneous data resources (UNIX FS, HPSS, UniTree, DBMS, etc.) distributed on multiple hosts and diverse platforms. The HDF is a file format and software library for storing all kinds of data, simple integers and floats or complex user-defined compound data types. The HDF employs a common data model with standard library APIs, providing efficient data storage and I/O access.

The HDF and the SRB offer valuable and complementary data management services, but they have not previously been integrated in an effective way. Earlier work had the SRB accessing HDF data either (a) by extracting entire HDF files, or (b) by extracting byte-streams through the SRB's POSIX interface. Approach (a) fails to take advantage of HDF's ability to offer interactive and efficient access to complex collections of objects. Approach (b) has been shown to be far too low-level to perform reasonably for some data extraction operations.

In discussions between NCSA and SDSC, it has been determined that a more effective approach is possible, one that uses modified HDF APIs on the server side to extract data from large files at the instruction of client-side HDF APIs and SRB as middleware to transfer data between the server and client. This approach would insert the HDF library and other object-level HDF-based libraries (such as HDF-EOS) between the SRB and a data storage source (such as a file system), making it possible to extract objects, rather than files or byte streams. Furthermore, these libraries typically offer query, subsetting, sub-sampling, and other object-level operations, so that these services might also be available..

2. Overview of SRB and HDF5

This section is a brief introduction of SRB and HDF5. For more information, you can visit the SRB and HDF5 websites at http://www.sdsc.edu/srb/ and http://hdf.ncsa.uiuc.edu/.

2.1. What is SRB

SRB is client-server middleware (or grid data software) that provides a uniform interface and authorization mechanism to access heterogeneous data resources (UNIX FS, HPSS, UniTree, DBMS, etc.) distributed on multiple hosts and multiple platforms. It is a distributed file system, a data grid management system, a digital library, and a semantic web.

* Sponsored by NLADR[1], NFS PACI[2] project in support of NCSA[3] and SDSC[4] Collaboration.

2.2. What is HDF5

HDF5 is a general-purpose library and file format for storing scientific data. At its lowest level, HDF5 is a physical file format for storing scientific data. At its highest level, HDF5 is a collection of utilities and applications for manipulating, viewing, and analyzing data in HDF5 files. Between these levels is the HDF5 software library that provides high-level APIs and a low-level data interface. HDF5 is a file format for storing all kinds of data and a library with standard APIs. It provides efficient data storage and I/O access and software and tools.

HDF5 can store two types of primary objects: datasets and groups. A dataset is essentially a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file. Using these basic objects, one can create and store almost any kind of scientific data structure, such as images, arrays of vectors, and structured and unstructured grids. You can also mix and match them in HDF5 files according to your needs.

3. The HDF-SRB model

We have designed a new mechanism, the HDF-SRB model to support object-level data access. The two basic requirements of the HDF-SRB model are simple and efficient. The HDF-SRB model has minimum changes the SRB code. It uses one set of objects for both server and client. It should have efficient data access by transferring only the required data (no redundant member object within an object) between client and server.

3.1. The HDF-SRB architecture

The HDF-SRB model consists of four basic components: the client (HDF client application or SRB client), the HDF-SRB module, SRB server, and the HDF library. Figure 1 illustrates the basic architecture of the HDF-SRB model.

Client applications are implemented using a set of APIs provided by SRB for sending requests and receiving responses to/from the SRB servers. The requests and responses are packed with HDF objects. The critical component is the HDF-SRB module, which connects the HDF clients to the HDF library on the server. The HDF-SRB module is responsible for packing and unpacking messages, or HDF objects, between the SRB and HDF components. The HDF library is installed with the SRB server for interactive access to HDF files on the server side.

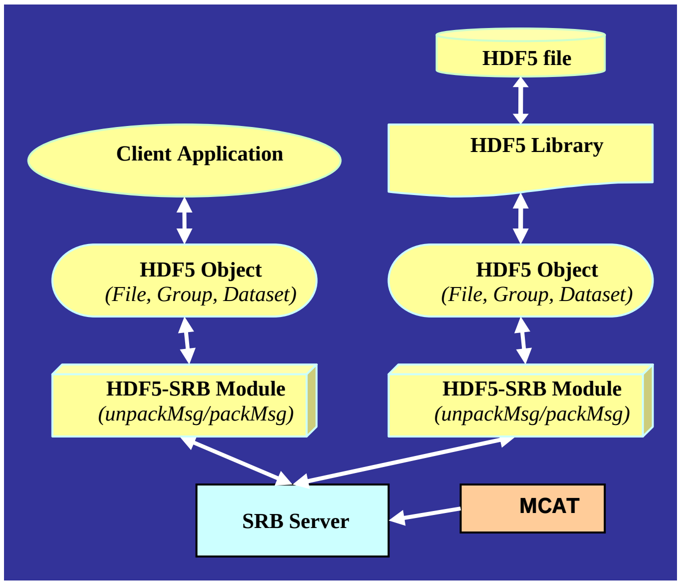


Figure 1 The HDF-SRB Model.

3.2. HDF data objects

In the HDF-SRB model, data objects are passed between the client and server rather than the entire file. There are several advantages for object level access. First, passing objects is more efficient than passing an entire file especially for large files. For example, if we want to access a small subset of a gigabyte dataset, we just bring the selected data to the client instead of the whole file. Second, it is easy to pass complex requests such as sub-setting. Because messages passing between the client and server are packed in data objects, there is no need to specify the format of the messages; messages, simple or complex, are self-explained in the object. Thirdly, it is easy to maintain the source code and extend to support new objects and new functions. Adding a new function to the object will not require any change to the data model.

There are three basic HDF5 objects (C structures): H5File, H5Dataset and H5Group. H5File is used to hold metadata about the file and the root group of the file. The file structure can be constructed by following the links that flow from the root group. H5Dataset contains data values and information about the data, such as data type and data space. H5Attribute is similar to H5Dataset, but contains user-defined metadata for groups and datasets. H5Datatype contains information about the data type of the dataset, such as data type class, order and size. H5Dataspace contains sizes of the dimensions of dataset. It is also used to calculate the data size and pass the subsetting information.

3.3. Client side API

h5ObjRequest() is the client side API, which is responsible for sending a client request to the server and for processing the response from the server. Request and response messages are packed in the HDF object structures.

3.4. Server side API

h5ObjProcess() is the server side API, which processes client request and sends results to back the client. The server side API does not call the HDF5 library directly. It calls unpackMsg() to construct the data object passed from the client. The data object then takes the operation and calls the HDF5 library.

3.5. Pack/Unpack routine enhancements

The packMsg() and unpackMsg() routines exchange structured data between client/server. A data structure is packed into a single byte stream before sending cross the network. Byte stream received is unpacked back into the data structure based on its definition. The enhanced packMsg() and unpackMsg() routines handle complicated data structures – string, pointers, pointers to arrays, arrays of pointers, etc.

3.6. General proxy functions

In SRB, proxy functions allow the execution of certain functions on the servers to improve performance. Examples of the use of proxy functions include data subsetting and filtering type operations where they can most efficiently be carried out on the servers where data reside.

To make it easier to implement and handle object-level HDF5 requests which can be quite complex, a new and more general SRB proxy function framework has been added. This framework can also be used by other developers to implement their own proxy functions.

In this framework, a client calls a new client API - srbGenProxyFunct() to make proxy request. Inputs for the srbGenProxyFunct function include:

- int functType - the type of proxy function. e.g., HDF5_OPR_TYPE
- void *inputStruct - Pointer to input struct.
- FormatDef inputFormat - packing instruction for inputStruct.
- void **outputStruct - Pointer to output struct.
- FormatDef outputFormat - packing instruction for outputStruct.

Inputs for the proxy function are given in an "inputStruct" which is a pointer to an arbitrary data struct and a "inputFormat" which is a character string containing the instruction for serializing the "inputStruct" into a single byte stream before sending across the netwotk to the server.

For example, an "inputStruct" may contain an "int" and a pointer to a string:

```
struct foo {
        int myIndex;
        char *myName;
};
```

The serializing instruction is a string containing "int myIndex; str *myName;" which instructs the serializing routine to treat the first member of the struct as an integer and the second member as a pointer to a string.

Similarly, the "outputStruct" and "outputFormat" specify the output and packing instruction for the output of the proxy function.

On the server side, the genProxyFuncEntries[] table defined in genProxyFunct.h is a switch table used by the server to determine the handling function for each proxy function type. Currently, the genProxyFuncEntries[] is defined as follows:

```
genProxyFunc_t genProxyFuncEntries[] = {
    {HDF5_OPR_TYPE, (func_ptr) h5ObjProcess},
};
```

The table contains only one entry, the HDF5 type (HDF5_OPR_TYPE) proxy function. The h5ObjProcess() function will be called to handle the HDF5_OPR_TYPE request. To implement a new type of proxy function, one needs to simply add one more entry to the genProxyFuncEntries[] table and a function to handle this type of request on the server.

4. Client application: the HDFView

The HDFView is a visual tool for browsing and editing NCSA HDF4 and HDF5 files. Using HDFView, you can

- view a file hierarchy in a tree structure
- create new file, add or delete groups and datasets
- view and modify the content of a dataset
- add, delete and modify attributes
- replace I/O and GUI components such as table view, image view and metadata view

For more details on HDFView, visit the NCSA HDFView webpage at http://hdf.ncsa.uiuc.edu/hdf-java-html/hdfview/index.html.

Supporting HDF-SRB in HDFView requires implementing HDF-SRB Java Native Interface(JNI) and adding new GUI components and data object

4.1. The HDF-SRB JNI

The HDF-SRB Java Native Interface (JNI) consists of an Java class and dynamically linked native library. The Java class declares static native methods, and the library contains C functions which implement the native methods. The C functions call the standard HDF-SRB client module.

The HDF-SRB JNI class contains only one native interface, h5ObjRequest(). h5ObjRequest () does two things: load the dynamic library and pass client requests to the C function.

public synchronized static native int h5ObjRequest (String srb_info[], Object obj, int obj_type) throws Exception;

The dynamic library (C implementation of the native interface) wraps the SRB client and converts data object between C and Java. When client calls the Java interface h5ObjRequest(), the dynamic library does the following tasks:

- Make connection to the SRB server
- Decode the Java object and construct C structure
- Send requests to the server in the form of C structure
- Encode server result in to Java object

4.2. The Java HDF-SRB objects

HDFView is implemented based on the HDF object package, a Java package which implements HDF4 and HDF5 data objects in an object-oriented form. The HDF Java object package provides a common standard Java API to access both HDF4 and HDF5 files. For more information on the HDF Object Package, visit http://hdf.ncsa.uiuc.edu/hdf-java-html/hdf-object/index.html.

To support HDF-SRB data objects, we have implemented the following Java package, ncsa.hdf.srb.obj, which contains:

- H5SrbFile extends FileFormat
- H5SrbGroup extends Group
- H5SrbScalarDS extends ScalarDS
- H5SrbCompoundDS extends CompoundDS
- H5SrbDatatype extends Datatype

These objects implement methods to deal with the client requests and data from the server. The native call, h5ObjReques(), passes the information through the objects. For example, the following code is how the we read data from remote file using H5SrbScalarDS::read().

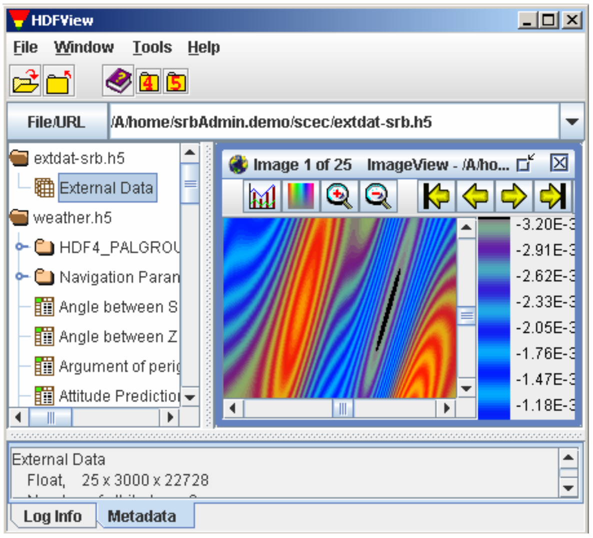
```
public Object read() throws Exception
{
    String srbInfo[] =
        ((H5SrbFile)getFileFormat()).getSrbInfo();
    if ( srbInfo == null  || srbInfo.length<5)
        return null;
    opID = H5DATASET_OP_READ;
    H5SRB.h5ObjRequest (srbInfo, this,
        H5SRB.H5OBJECT_DATASET);
    return data;
}
```

4.3. The GUI components

Since HDFView is built on modular fashion, the GUI components are transparent to data access. There is not much change to the GUI components. We added SRBFileDialog class to the GUI. SRBFileDialog class is used to make server connection by using the Java API for Real Grids On Networks (JARGON). JARGON is a pure java API for developing programs with a data grid interface. The API currently handles file I/O for local and SRB file systems, as well as querying and modify SRB metadata. For more information on JARGON, read http://www.sdsc.edu/srb/jargon/index.html.

Figure 2 shows two examples of accessing HDF5 files on the SRB server in HDFView. The first file, extdat-srb.h5, contains one dataset of size about seven gigabytes (25*3000*22728*4). With SRB support, we can have instant access to subset of the seven gigabyte dataset. Without SRB support, it would take hours to transfer the whole file to local machine.

The second example, weather.h5, shows how we can have instant access to the file structure. With SRB support, we can browse through the file structure without bringing the content of the file to local machine.



External Data
Float, 25 x 3000 x 22728

Figure 2 The SRB File Access in HDFView

4.4. Conclusions

The HDF-SRB model integrates two successful technologies, the SDSC SRB and the NCSA HDF, to create a new, object-lvel distributed data service. The SRB serves as standard middleware to transfer data between the server and client. HDF5 provides interactive and efficient access to datasets or subsets of datasets in large files without bringing entire files into local machines. A new set of data structures and APIs have been implemented to the SRB support such object-level data access.

A working prototype of the HDF5-SRB data system has been developed and tested. The HDF-SRB has been proved to be very efficient in large files. We have implemented SRB support in HDFView. Using HDFView, we can have instant access to file structures and fast access to subset of large dataset. Without HDF-SRB support, it might take hours to bring the file to a local machine.

This project has been a very successful team effort between SDSC and NCSA. Both SRB and HDF5 are very complex and the implementation of such a server/client system requires a full understanding of the two technologies. The SDSC SRB and NCSA HDF teams have worked together on all parts of the project, including designing, coding and testing.

References

[1] http://www.nladr.org
[2] http://www.npaci.edu/
[3] http://www.ncsa.uiuc.edu/
[4] http://www.sdsc.edu/
[5] http://www.sdsc.edu/srb/
[6] http://hdf.ncsa.uiuc.edu/